Diversifying Big Data in Parallel

A Thesis

submitted to the designated by the
General Assembly of Special Composition
of the Department of Computer Science and Engineering
Examination Committee

by

Konstantinos Noulis

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
WITH SPECIALIZATION
IN SOFTWARE

University of Ioannina
February 2017

DEDICATION

Dedicated to my grandfather Kostas Noulis.

Αφιερωμένο στον παππού μου Κώστα Νούλη.

ACKNOWLEDGMENTS

I shall express my gratitude to my supervisor, Professor Evaggelia Pitoura, for believing in me and giving me the opportunity to broaden my knowledge.

I would also like to thank my family and partner in life, Periclis, Aristea, Marina and Eleni as well as my friends George, Efi and Babis for supporting me in every step of my life. Special thanks, for all the great times we had, to my new friends Fanis and Maria, without whom my academic journey would be poorer.

TABLE OF CONTENTS

List of Tables

List of Figures

ABSTRACT

Konstantinos Noulis

MSc, Department of Computer Science and Engineering

University of Ioannina, Greece

February 2017

Diversifying Big Data in Parallel

Supervisor: Evaggelia Pitoura

A characteristic that makes a query result distinctive is apparently quality. One way of enhancing the quality of a query result is diversification. Many ways have been already proposed in order to diversify a query result. In this study we utilize the algorithm of Dissimilarity and Coverage as well as the Max Cover algorithm in an attempt to diversify a query result, which is substituted by several sets of data. The novelty that is introduced in this study is the perspective through which we approach those algorithms. Diversification is conducted with the use of parallel implementations of the aforementioned algorithms in a distributed environment. With this thesis, we intend to propose a new way of diversifying Big Data efficiently, taking at the same time advantage of new distributed platforms.

Εκτεταμενη Περιληψη στα Ελληνικα

Κωνσταντίνος Νούλης του Περικλή και της Αριστέας
ΜΔΕ, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
Φεβρουάριος 2017
Παράλληλη υλοποίηση αλγορίθμων ποικιλομορφίας με βάση τη διαφορετικότητα σε μεγάλα σύνολα δεδομένων
Επιβλέπουσα: Ευαγγελία Πιτουρά

Μια αναζήτηση ενός χρήστη σε μια βάση δεδομένων κρίνεται από το αν και κατά πόσο το αποτέλεσμα αυτής της αναζήτησης θα βοηθήσει το χρήστη να βρει γρήγορα και εύκολα αυτό που ψάχνει. Η ποικιλομορφία των αποτελεσμάτων που παρουσιάζονται μετά από μια αναζήτηση σε μια βάση δεδομένων είναι κάτι που έχει αυξανόμενο επιστημονικό ενδιαφέρον τελευταία. Για να γίνει λοιπόν πιο ποιοτικό και ελκυστικό ένα αποτέλεσμα αναζήτησης επιδέχεται κάποιας επεξεργασίας, πριν την τελική παρουσίαση του στο χρήστη. Η επεξεργασία αυτή καθιστά το αποτέλεσμα της αναζήτησης πιο αξιόπιστο, αφού παρουσιάζει ένα υποσύνολο του αρχικού αποτελέσματος, το οποίο παραθέτει αντιπροσωπευτικά δείγματα και είναι επίσης τόσο διαφοροποιημένο, ώστε τα αποτελέσματα που θα εμφανιστούν στο χρήστη να είναι ποικιλόμορφα και εύστοχα.

Μέχρι στιγμής έχουν γίνει διάφορες απόπειρες για τη σχεδίαση ενός μοντέλου που θα παράγει αποτελέσματα με τις παραπάνω ιδιότητες. Κάθε απόπειρα θέτει ένα διαφορετικό ορισμό για τη ποικιλομορφία των δεδομένων, κοιτάζοντας το πρόβλημα από διαφορετική οπτική γωνία. Σε αυτή την εργασία μελετάται ο αλγόριθμος [8] του DisC Diversity. Ανάμεσα στις διάφορες θεωρήσεις, επιλέξαμε να μελετήσουμε τον αλγόριθμο DisC διότι καλύπτει καλύτερα τις ανάγκες του προβλήματος που έχουμε θέσει, επιστρέφοντας στο χρήστη αποτελέσματα που είναι διαφοροποιημένα μεταξύ τους και την ίδια στιγμή καλύπτουν όλο το σύνολο των αποτελεσμάτων του ερωτήματος στη βάση. Η προσέγγιση που γίνεται για την επίλυση του προβλήματος με αυτό τον τρόπο εμπεριέχει την μετατροπή

του DisC αλγορίθμου σε τρεις παραλλαγές, που επιλέγουν αποτελέσματα για το τελικό σύνολο με βάση τρία διαφορετικά κριτήρια.

Εκτός από τον παραπάνω αλγόριθμο, γίνεται προσπάθεια προσέγγισης του προβλήματος με μια παραλλαγή του αλγορίθμου [20] που λύνει το Maximum Coverage πρόβλημα, το οποίο είναι NP-hard, όπως και το αρχικό πρόβλημα αυτής της μελέτης. Με αυτή τη μέθοδο, στόχος είναι να δημιουργηθεί μια κατάταξη των αποτελεσμάτων που επέστρεψε το ερώτημα. Αυτή η κατάταξη πρέπει να είναι τέτοια ώστε οσαδήποτε αποτελέσματα επιλεγούν να παρουσιαστούν στο χρήστη, να καλύπτουν σε κάθε περίπτωση σε μέγιστο βαθμό το σύνολο με τα αποτελέσματα του αρχικού ερωτήματος. Η κάλυψη σε αυτό το σημείο έγκειται στην ομοιότητα των αποτελεσμάτων σε σχέση με τα υπόλοιπα.

Η καινοτομία που εισάγει αυτή η εργασία είναι η υλοποίηση των παραπάνω αλγορίθμων, και ειδικά του DisC Diversity, με τέτοιο τρόπο ώστε να μπορούν να εκτελεστούν παράλληλα σε ένα κατανεμημένο περιβάλλον. Με αυτό τον τρόπο θέλουμε να πετύχουμε το ζητούμενο της εργασίας, που είναι η επεξεργασία μεγάλου όγκου δεδομένων. Για αυτό το σκοπό χρησιμοποιούμε τη λογική του Map Reduce, που μπορεί να εφαρμοστεί μέσω της πλατφόρμας του Hadoop. Το Hadoop αποτελείται από ένα τρόπο δομής και διαχείρισης ενός αρχείου συστήματος και ένα τρόπο επικοινωνίας μεταξύ διασυνδεδεμένων υπολογιστών, ώστε να μπορεί να εκτελεστεί πάνω του ένα πρόγραμμα που έχει τη λογική του Map Reduce. Το Hadoop είναι λογισμικό ανοιχτού κώδικα και έχει δημιουργηθεί από την Apache, βασισμένο σε μια ερευνητική εργασία της Google η οποία περιγράφει το Map Reduce και το Google File System, που δημιουργήθηκαν στα εργαστήριά της.

Για τα πειράματά μας χρησιμοποιούμε τεχνητά αλλά και πραγματικά σύνολα δεδομένων, που προσομοιώνουν σύνολα αποτελεσμάτων που έχουν επιστραφεί από ερωτήματα σε κάποια βάση δεδομένων. Τα τεχνητά σύνολα είναι κατασκευασμένα με τέτοιο τρόπο ώστε να αντιπροσωπεύουν δύο μεγάλες κατηγορίες. Η διαφοροποίησή τους έγκειται στην κατανομή των δεδομένων όσον αφορά τα χαρακτηριστικά τους. Υπάρχουν σύνολα δεδομένων που είναι φτιαγμένα με τυχαίο τρόπο αλλά και άλλα που περιέχουν λογικές ομάδες αποτελεσμάτων που έχουν πιο σχετικά χαρακτηριστικά σε σχέση με τα υπόλοιπα, δημιουργώντας συστάδες. Πάνω σε αυτή τη διαφοροποίηση γίνεται μια επιπλέον μελέτη για το αν και κατά πόσον ένας διαφορετικός διαχωρισμός των δεδομένων πριν την εκτέλεση του βασικού αλγορίθμου μπορεί να οδηγήσει σε καλύτερα αποτελέσματα. Ο διαχωρισμός που επιχειρείται γίνεται με την χρήση του αλγορίθμου συσταδοποίησης k-means.

Η επιτυχία των αλγορίθμων κρίνεται από το ποσοστό ευστοχίας του συνόλου δεδομένων που παράγουν ως αποτέλεσμα. Η ευστοχία ή αστοχία του τελικού συνόλου δεδομένων υπολογίζεται με βάση το ποσοστό των λανθασμένων αποτελεσμάτων που περιλαμβάνονται σε αυτό. Ένα αποτέλεσμα χαρακτηρίζεται ως λανθασμένο, αν έχει όμοια χαρακτηριστικά με κάποιο άλλο που βρίσκεται και αυτό μέσα στο τελικό σύνολο, που θα παρουσιαστεί στο χρήστη. Αυτό το αποτέλεσμα είναι περιττό και αποτελεί αστοχία του αλγορίθμου.

Η αξιολόγηση των αποτελεσμάτων βασίζεται κυρίως στο χρόνο εκτέλεσης των αλγορίθμων αλλά και στην απόδοσή τους, η οποία κρίνεται από το μέγεθος του τελικού συνόλου δεδομένων που παράγεται σε σχέση με το αρχικό αλλά και από το πόσο εύστοχο είναι. Σε πρώτη φάση γίνεται μια ξεχωριστή αξιολόγηση των αλγορίθμων πάνω στην ίδια πειραματική βάση, ενώ ακολουθεί μια σύγκριση των δύο κατηγοριών αλγορίθμων. Στο στάδιο της σύγκρισης παρατηρείται η επίπτωση που έχουν διάφορες παράμετροι που έχουμε θέσει στην απόδοση των αλγορίθμων. Οι παράμετροι αυτοί είναι η πληθικότητα του αρχικού συνόλου δεδομένων, η δομή και η κατανομή του αρχικού συνόλου δεδομένων, ο αριθμός των Mappers και Reducers που χρησιμοποιούνται για την εκτέλεση των πειραμάτων, όπως και η μετρική που έχουμε θέσει για τον υπολογισμό της ομοιότητας μεταξύ δύο αποτελεσμάτων.

Μετά από την εκτέλεση των παραπάνω πειραμάτων και την επεξεργασία των αποτελεσμάτων τους με βάση τα κριτήρια αξιολόγησης που έχουμε θέσει, κρίνουμε αν και κατά πόσον κάποιος αλγόριθμος προτιμάται, ώστε να εφαρμοστεί για την επεξεργασία ενός εκτεταμένου συνόλου δεδομένων, με σκοπό να παρουσιαστεί στο χρήστη ένα αρκετά ποικιλόμορφο, διαφοροποιημένο και πληροφοριακό σύνολο αποτελεσμάτων.

CHAPTER 1

INTRODUCTION

1.1 Thesis Scope

Data mining is a field of computer science with ever growing scientific interest and is already applicable in everyday life. For example, extracting information out of stored data can be precious for enterprises in order to better understand the willing of the consumers. Moreover, trends can be created in social media based on the information retrieved by the activities of the users or even health organizations can be aided in their research.

The amount of data that has to be processed though has massively grown in the last years. A vast amount of information is stored and can provide valuable knowledge if processed. "Big Data" is a new term which describes a holistic information management strategy that includes and integrates many new types of data and data management alongside traditional data. Big data has also been defined by the four "V"s as described in [1].

- Volume, which stands for the scale of the data. It is estimated that every day 2.5 trillion gigabytes are created and that 40 zettabytes of data will be stored by 2020.
- Velocity, which stands for the need to analyze streaming data. Online trade information processing or sensor-based information reaction could depict this parameter.

- Variety, which describes the different forms of data that are stored. From posts on social media, to videos on youtube and from information from wearables to scientific papers, there exist a huge amount of different kinds of data.
- Veracity, which expresses the uncertainty of the stored data. The inaccuracy of the data can conclude to wrong decisions and cost money and effort.

Processing large scale datasets with traditional algorithms cannot be accomplished or is extremely tough regarding the computational power that is needed. Novel algorithms designed with a parallel logic are developed and executed on distributed systems and clusters of commodity hardware in order to overcome this obstacle. Thus, execution time and efficiency is enhanced while using everyday machines with regular characteristics and without spending a fortune in boosting a supercomputer with extreme possibilities. Database processing has been expanded and is conducted with other ways too, e.g. noSQL systems.

Google has a leading role in this sector, since its file system has been specially designed to process large scale data, but other platforms with free software exist as well, which are equally efficient and open to public. One such platform, which is going to be explained in details later on, is called Hadoop and is developed by Apache. Hadoop has been adopted by key role players, like Yahoo, Amazon, Adobe, Twitter, etc and Hadoop-related products and services generate more than $800 million in revenue [19]. Hadoop has also been used in scientific research. For example, it has advanced the research for cancer [2] in Arizona State University.

The algorithms that are used for large scale datasets processing need to tackle all the four “V” issues that define Big Data. What makes the stored data valuable though is the quantity and quality of the information that can be retrieved out of them. Result diversification has attracted a lot of attention lately as a means of enhancing the quality of the query results that are retrieved of a storage system. Results with higher quality are more applicable and lead to a more efficient processing of large scale datasets. This issue constituted our motivation for this study, which aims in enhancing the quality of the query results when processing large scale data. Inspiration was also derived by studies that use Map-Reduce framework in order to pursue a deeper analysis of the query results [20].

Many ways of diversifying data have been proposed so far, viewing the problem of diversification either based on novelty, or new content or even coverage. DisC Diversity algorithm [6] makes a fair trade on the dissimilarity of the resulting objects and the coverage of the initial dataset. However, the size of the initial dataset that can be handled by DisC is limited. With a non-parallel implementation of the algorithm, as the input size gets larger the efficiency drops and the execution time increases.

The goal of this study is firstly to propose a parallel implementation of the DisC Diversity algorithm, which could handle large scale datasets efficiently. A second algorithm is also taken under consideration as a means of diversifying large scale datasets. This is performed through a parallel implementation of the Max Cover algorithm, which selects a number of sets of objects that cover at the most the initial dataset.

The distributed environment that hosted our experiments was the cluster of commodity hardware that exists in our school. The experiments were conducted against synthetic and real datasets as well. The synthetic datasets hold either one a different structure with different distribution regarding the containing objects. Experimental evaluation has been made with different parameters, regarding the input dataset size or structure, the number of mappers and reducers or the metric that is used to define similarity.

The outcome of this study is a complete evaluation of the proposed algorithms in regards to their execution time and efficiency. Efficiency is calculated based on the cardinality and the accuracy of the resulting dataset. Accuracy of the resulting dataset is judged based on the error objects that it contains, i.e objects of the resulting dataset that are similar to others. Furthermore, a comparison of the algorithms shows the impact of the experimental parameters in their execution and efficiency and helps to distinguish if one of them should be prefered in order to diversify large scale data. Finally, an extra study is made in order to specify if a differentiation in the partitioning of the initial datasets could lead to different results. Partitioning is either performed in a random way or by applying a clustering method before the actual run of the algorithm.

This thesis contains a first approach of solving the diversification problem with a parallel implementation. Both DisC Diversity and Max Cover are NP-hard problems, so the solution that we propose is a greedy one. Our goal is to investigate and propose one algorithm that meets our expectations and fulfills the criteria we have set.

1.2 Thesis Structure

The structure of this thesis is as follows. In Chapter 2, we describe the fundamentals of the Hadoop file system and the Map Reduce logic. Moreover, we refer to related work regarding the data diversification and the maximum coverage problems. In Chapter 3, we present the algorithms that constitute the backbone of this work. Subsequently, we illustrate the way that we changed those algorithms in order to be able to run in a distributed environment. Within Chapter 4, the setup of the environment, that the algorithms were tested, is explained, along with more details on the datasets, that were used, and the evaluation criteria, that we took under consideration. In Chapter 5, the results of this study are shown and finally, in Chapter 6, the conclusions are summed up and some future work is mentioned.

CHAPTER 2

RELATED WORK AND THEORETICAL BACKGROUND

2.1 Data Diversification

Result diversification has attracted a lot of attention as a means of enhancing the quality of query results presented to users (e.g. [3]). Query results with enhanced quality are highly appreciated in many cases with range from applications that cooperate with a database system to a simple search on the internet, where the user should run into an interesting result quickly and effortlessly. In addition, a more enhanced and qualitative way of producing results out of queries could help in regards to the personalization of the query results, since data mining algorithms are often used in order to make it easier for the user to search in the internet. A query result should ideally be a subset of the original dataset which is as explanatory and as diverse as possible. As an example, consider a consumer who wants to buy a tablet and makes a query at google about it. The desirable outcome of this query should be a diverse subset of tablet choices. A diverse result, which contains various brands and models with different screen sizes and other technical characteristics is intuitively more informative than a homogeneous result containing only tablets with similar features.

Generally, the diversification problem is defined as follows. Given a set *P* of *N* items available and a restriction *K* on the number of desired resulting items, select a subset *S* of *K* items out of the *N* available, such that the diversity among the items of *S* is maximized. Until now, various approaches have been made in order to define diversity and try to solve the problem. The metrics that have been proposed so far mainly exploit:

- content. The result set contains objects, all of which are dissimilar to each other (e.g. [3]). This way, diversity is interpreted as an instance of the p-dispersion problem. That is to select p out of n given items, such that the minimum distance between any pair of selected items is maximized. This method can find usage in locating facilities, that should be dispersed, e.g. franchise branches or nuclear power plants.
- novelty. The result set contains objects that provide new information when compared to what was previously presented (e.g. [4]). Novelty could be correlated with diversity, with a slight difference in the perspective of the information retrieval concept, i.e novelty-based systems avoid redundancy whereas diversity deal with ambiguity.
- semantic coverage. The result set contains objects that belong to different categories or topics (e.g. [5]). This approach contains creation of query concepts out of the original queries, perhaps clustering of queries and use of probabilistic functions in order to select the results.

Our study has focused on a novel way of diversifying data, which was firstly introduced in [6]. This method is called DisC Diversity, which stands for diversification based on dissimilarity and coverage. In a nutshell, this method performs a diversification on the resulting set of a query, securing that the final resultset, that it outputs, satisfies two restrictions. It is a subset of the original resulting set with its elements being as different as possible and at the same time covering as much as possible the original resulting set. Following the previous example, by applying this algorithm on the original resultset of all possible tablets that a consumer could buy, a representative subset shall contain tablets that cover any possible choice and have distinctive characteristics. This is an example of quality enhancement in query results, which implies also a quality enhancement in the information that is retrieved out of the stored data.

DisC Diversity algorithm though present lower efficiency levels when processing a large scale dataset, especially if implemented in a centralized manner. The goal of this thesis is to implement DisC Diversity algorithm in a distributed way in order to be able to manipulate larger inputs, without affecting the efficiency of the algorithm and maintaining at the same time low levels on the execution time. More details on the original algorithm and our perspective of transforming it in a distributed way are given later on in Chapter 3.

2.2 Maximum Coverage Problem

The maximum coverage problem is a trivial question in computer science. As input for this problem we consider several sets and a number *k*, taken that the sets may have some elements in common. The objective is to select at most *k* of these sets, such that the maximum number of the elements are covered, i.e. the union of the selected sets has maximal size. Formally the definition of the maximum coverage problem is as follows. Given a number *k* and a collection of sets $S = S1, S2, \dots , Sm$, find a subset $S' \subseteq S$ of sets, such that $|S'| \leq k$ and the number of uncovered elements $|\cup_{Si \in S'} S_i|$ is maximized.

Maximum coverage problems are faced frequently enough. Suppose that a search engine wishes to focus its attention to one thousand queries such that as many users as possible will see one of them. Selecting these queries is a maximum coverage problem. Problems like these arise when we seek the best possible collection of items but the items may overlap in the value they provide, and should not be double-counted. Other situations that the usage of this algorithm is demanded could be the selecting of places where a set of charity boxes could be installed in order to be available to as many people as possible or even making a selection of a set of websites where an advertisement banner can be placed in order to be seen by the majority. Other unexpected usages could be in other science fields as well, like for example the collection of five vitamins that wards off the most ailments.

Both diversification and maximum coverage are NP-complete problems. Optimization problems like these are solved with approximation algorithms that find approximate solutions.

Approximation algorithms are often associated with NP-hard problems, since it is unlikely that there can ever be efficient polynomial-time exact algorithms solving them. Unlike heuristics, which usually only find reasonably valid solutions rather fast, one wants provable solution quality and provable run-time bounds. Ideally, the approximation is optimal up to a small constant factor, for instance within 5% of the optimal solution.

Many alternatives of the Max Cover algorithm have been proposed, such as the weighted version or the budgeted maximum coverage. The most commonly used though is the greedy algorithm. As any other greedy algorithm, the main idea of this maximum coverage algorithm is to choose every time sets to include to the output, which contain the largest number of uncovered elements.

Nonetheless, max-cover problems arise frequently at large scale. Dasgupta et al., for instance, employ this formulation in pursuit of discovering fresh content for web crawler scheduling policies, i.e. finding as much new content as possible by crawling at most k webpages. Saha and Getoor used the Max Cover formulation for a multi-topic blog-watch application, i.e. finding at most k blogs to read interesting articles on a list of topics. Although greedy algorithm is known to be the best possible polynomial time approximation, it requires an updating of the scores of arbitrary elements after each step, and hence becomes intractable for large datasets. Following in this study, a parallel implementation of the maximum coverage algorithm is taken under consideration, which has almost the same approximation as the greedy one. More details on this are given in Chapter 3.

2.3 Distributed Computing

Distributed computing is a field of computer science that deals with distributed systems. In a distributed system, components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components.

On the same side lie programming methods that use messaging. Message Passing Interface (MPI) is a standardized and portable message-passing system designed approximately in 1995 by a group of researchers from academia and industry to function on a wide variety of parallel computers. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in C, C++, fortran, etc. There are several well-tested and efficient implementations of MPI, many of which are open-source or in the public domain. These fostered the development of a parallel software industry, and encouraged development of portable and scalable large scale parallel applications.

2.3.1 Map Reduce Programming Model

Basic Concept

The conception of message passing interface was evolved later on by Google. Six years after its foundation, in 2004 Google introduced the Map-Reduce framework [7], which along with other four products (BigTable, Borg, Chubby and Dremel) made the company a trademark in terms of searching and querying data among other. It is not out of luck the phrase “google it”, ”that everyone uses today.

Map-Reduce is a rather simple concept that can be applied for processing large scale datasets fast and effectively. It is a simple and powerful interface that enables automatic parallelization and distribution of large scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity machines. It is used in computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a specific day. The computation takes as input key/value pairs and is expressed in two functions:

- The Map function, which is provided by user. It takes as input the key/value pairs and provides an intermediate pair of key/value. The Map-Reduce library groups together all the intermediate values that are associated with a specific key and passes them to the next step.

- The Reduce step, which is also written by user. Its input is an intermediate key along with a set of values that are associated with it. Normally, it merges these values to a smaller set. Typically no or one value is produced by this step. The intermediate values are passed to this step by an iterator. This allows system to handle list of values that are too large for the memory. The reduce step normally precedes a Shuffle step, which makes all data that belong to one key to be located on the same machine.

A common example that depicts the Map-Reduce logic is the “Word Count Example”, which is a good first attempt with the Map-Reduce framework. In order to count the appearances of every word in a large text, the following two functions could be used.

```
map(String key,String value) {
  // key: document name
  // value: document contents
  for each word w in value:
     Emit_Intermediate(w, "1");
}
```

```
reduce(String key,Iterator values){
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
}
```

Algorithm 2.1: Word Count with MapReduce

The map function emits every word of the document along with its occurrence, which is set to one for this example. Then reduce function gathers the list of ones for every word and sums them up. The result is a key/value pair, that contains every word in the document along with its occurrences. User is responsible for writing the map and reduce function correctly. A MapReduce library, that is installed in the background, is meant to handle the transfer of the data, the passing of the values in the iterator and the communication between the interfaces.

Execution Flow

Map-Reduce is mainly executed over clusters of commodity hardware. The official terminology of the machines of the cluster is “master” and “worker”. There exists one master that coordinates the execution of the program and many workers that pull out the actual work. The initial dataset is splitted into M parts which are fed to the Mappers. The intermediate data is splitted to R pieces using a partitioning function (e.g. hash(key) mod R) and are transferred to the Reducers. The number of M and R pieces is pre-defined by user.

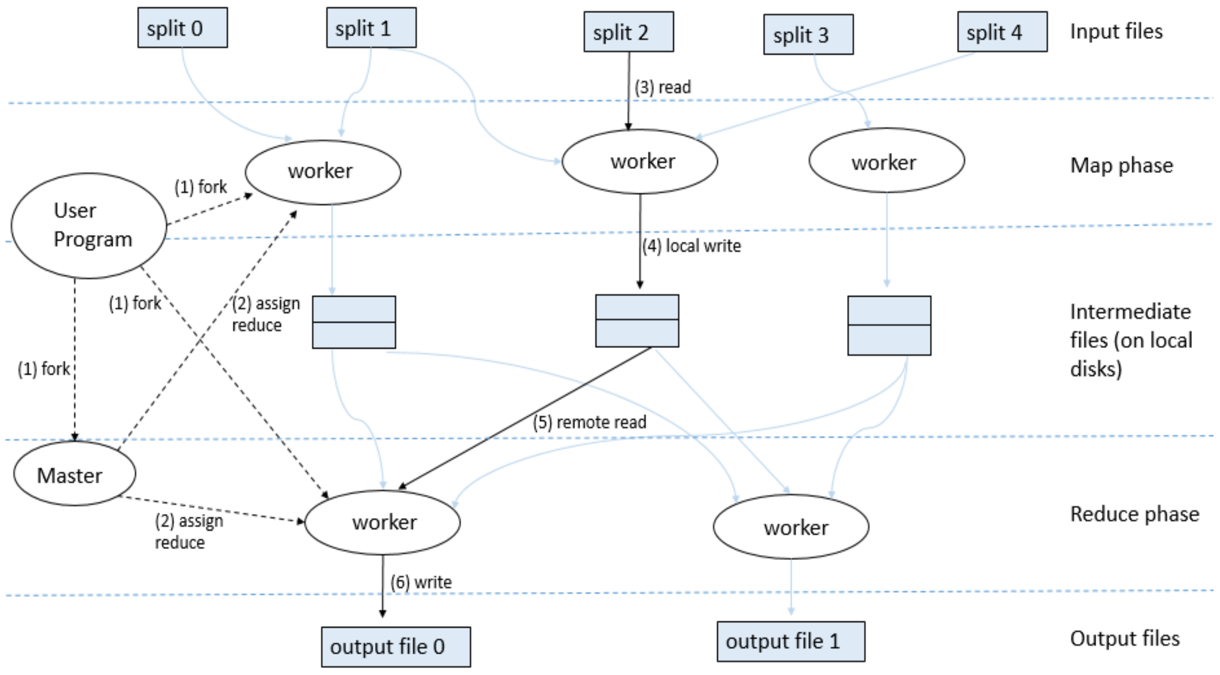


Figure 2.1: Map Reduce Execution Flow [7]

When a Map-Reduce program is called the following sequence of actions occurs. The numbering of the steps is reflected at Figure 2.1.

1. The MapReduce library firstly splits the dataset in M partitions of a size pre-controlled by the user. It then starts a copy of the program in all the machines of the cluster.
2. The master assigns either a map or a reduce task to a worker, when the later is idle.
3. Every worker reads data from a corresponding input split. It then parses key/value pairs, applies the map logic and stores the produced intermediate key/value pairs in a memory buffer.
4. Periodically the buffered pairs are stored at local disc partitioned into R regions with the partitioning method. The locations of the stored key/value pairs are passed back to the master, who is responsible of forwarding these locations to the reducers.

5. The reducer uses remote procedure calls to read the intermediate data, which is stored in the local discs of the mappers. Then it sorts the data based on the key, so as all occurrences of the same key to be grouped together.
6. The reducer iterates all same intermediate keys and passes all the corresponding values to the reduce function.
7. When all mappers and reducers have ended their work, the master wakes up the caller program and returns.

After a successful execution of a MapReduce program, the resulting key/value pairs are stored in R files, with names predefined by the user. All these results could be then gathered in a final file. More often though user keeps the output files as such, because they are meant to be an input for a next Map-Reduce execution.

Fault Tolerance

When dealing with large scale data, that are being processed on top of a distributed system, it is crucial for the system to be tolerant in failures. Indeed, Map Reduce framework handles faulty behaviors gracefully. The main benefits of the standard approach for fault tolerance, implemented in Hadoop, consists on its simplicity and that it seems to work well in local clusters. In order to be prepared for a failure, the master node stores periodically a checkpoint, i.e. a complete image of all the data structures that are being processed. When the process exits abnormally, an exact copy of the data is provided to the master in order to continue from the last checkpoint. On the other hand, failures in worker nodes are prohibited by a polling mechanism from the master node. The master pings every worker node periodically in order to verify if it is alive. If no response is received in a certain amount of time the worker node is considered as dead. Any map tasks that were in progress by the specific worker return to the initial “idle” state and therefore become eligible for scheduling on other workers. The same applies to completed tasks. These tasks also are handled by another worker as their intermediate results were stored inside the disc of the faulty worker. Retrieving the data from this faulty node is not feasible.

Additional Features

Although the basic functionality of the Map Reduce library is fair enough for processing scaled datasets, a set of extensions make it more useful. Some of these features, that boost the efficiency and user-experience of the Map Reduce framework, are the following:

- A library that provides a counter facility to count occurrences of various events.
- A web interface that contains status information about the running jobs. The master node runs an internal HTTP server and exports a set of valuable data for the user.
- Map-Reduce jobs can be executed in a local environment for testing purposes.
- The Map-Reduce library provides support for reading input data in different formats. For example the "Text" mode input treats each line as a key/value pair. The key could be the offset in the file and the value could be the contents of the line.
- The partitioning function that is responsible for splitting the intermediate data could be changed in order to satisfy the user's needs. For example, sometimes it could be useful to feed another argument other than the key to the hash function.
- Last but not least is the Combiner function. In some cases there is a really significant and meaningless repetition of intermediate key/value messages. In order to avoid this, user is allowed to specify an optional Combiner function that applies a partial merging of the data before they are sent to the reducer. A good example for this is the "Word Count Example" which is described above. The mappers will produce a ton of <key, 1> messages, which will flood the network. Moreover, the reducers will be overloaded with receiving and merging this data. A combiner function could be applied within the mapper in order to sum up the occurrences of every key before sending the key/value pair to the network. The Combiner function works exactly as the reducer, with the difference that it produces an intermediate key/value pair in contrast with the reducer which creates the output.

2.3.2 Hadoop Framework

The aforementioned Map Reduce programming model is applicable only if it runs over a specific framework. One such is the Apache Hadoop framework, which is an open-source software framework for distributed storage and distributed processing of very large datasets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework. The core of Apache Hadoop consists of the processing part, which is called MapReduce and the storage part, known as Hadoop Distributed File System (HDFS). The genesis of Hadoop came from the Google File System paper [8], that was published in October 2003. The name "Hadoop" was given by its creator Doug Cutting's son, after a toy elephant.

The Google File System

The so-called "Google File System" is a specialized file system, designed inside the labs of Google after many years of observation and stress-testing of traditional file systems. It is implemented in order to meet the rapidly growing demands of data processing. The file system consists of thousands of storage machines built from inexpensive commodity parts. The quantity and quality of the components virtually guarantee that some will fail probably instantly and some other will fail and not recover at all. This fact led to the concept of this file system, which is very tolerant to faults, as it is extremely difficult and expensive to maintain such amounts of hardware providing a high level of quality of service at the same time.

GFS Architecture

A GFS cluster mainly consists of a single master and multiple chunkservers and is being accessed by multiple clients. Each of these is a typical commodity Linux machine that runs a user-level server process. It is equally possible to run a chunkserver or a client process on any machine of the cluster. Data is maintained in big chunks, typically in size of 64MB, inside the local discs of the underlying Linux. The design delivers high aggregate throughput to many concurrent readers and writers performing a variety of tasks. This is achieved by separating file system control, which passes through the master, from data transfer, which passes directly between chunkservers and clients.

The GFS architecture is illustrated in Figure 2.2. In a nutshell, the master maintains all the file system metadata (namespace, access control information, the mapping from files to chunks, and the current locations of chunks). It also controls system-wide activities such as chunk-lease management, garbage collection of orphaned chunks, and chunk-migration between chunkservers. The master periodically communicates with each chunkserver in heartbeat messages to give it instructions and collect its state. On the other hand, the client code, which is linked into each application, implements the file system API and communicates with the master and chunkservers to read or write data on behalf of the application. Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunkservers. Neither the client nor the chunkserver caches file data.

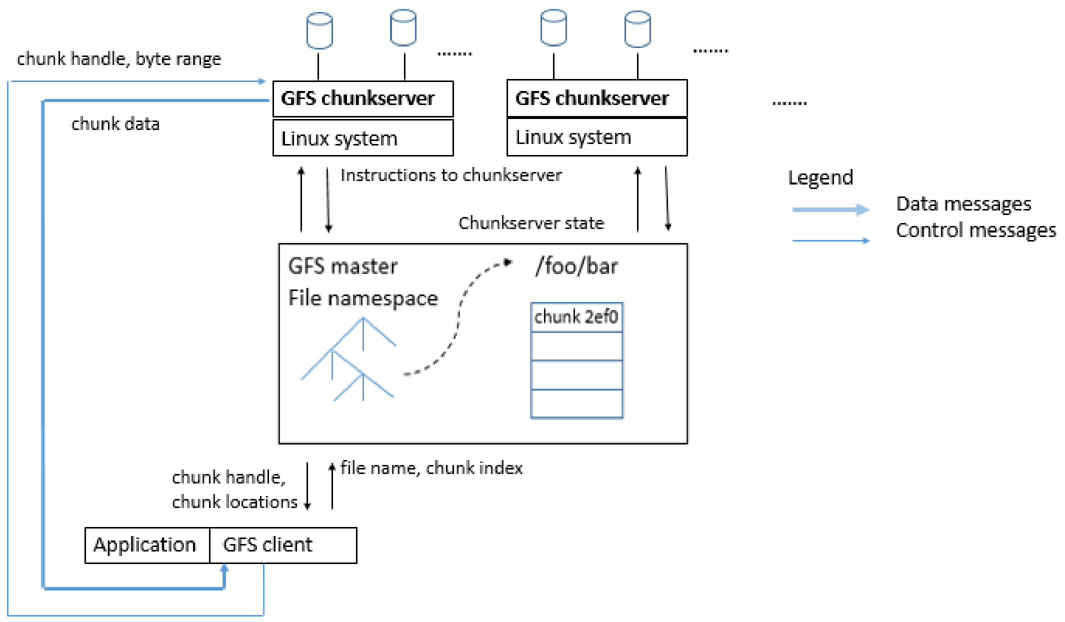


Figure 2.2: The Google File System Architecture [8]

The truly innovative of this file system is the separation of the control and the data transfer. This prevents the master of becoming the bottleneck for the system. The clients do not actually read or write file data through the master. Instead, a client requests from the master which chunkservers it should contact. It caches this information for a limited time and interacts with the chunkservers directly for any subsequent operations needed. Same architecture is followed by Apache Hadoop File System, which was created with reverse engineering, based on GFS.

Specifications and Features

The chunk size is by default set to 64 bytes, a number much larger than a typical file system block size. The advantage that is derived out of this is firstly the reduced communication between master and clients. Any information that is included inside a chunk requires only one client request. Moreover, due to the fact that many operations could be held inside one chunk, the network overhead can be reduced, as a persistent TCP connection will be kept over to the chunkserver. The operations that are supported by the GFS are the usual for create, delete, open, close, read and write files. On top of those, GFS has also snapshot and record append operations. File namespace mutations (e.g. file creation) are atomic. No conflict is permitted as they are handled exclusively by the master.

What makes this file system different is the fault tolerance level and the high availability that is provided. This is accomplished with many ways, one of which is replication. Master is responsible for replicating data at a number that is pre-configured by the user. The storage target is decided in a way that the replicas would be safe in case of a failure, e.g. replicas are stored in different racks. Furthermore, the file system has its own recovery mechanism in case of an erroneous situation. Master keeps an image of the system periodically by taking a snapshot. All this information, which specifies the chunk mapping, the client states, the master's next step, etc along with the subsequent logs help the system recover and continue working as previously. Last but not least, GFS provides high quality side applications, as the garbage collector. In fact, this works in a different way than in other file systems. For any file that is deleted from application, the master logs the deletion immediately. The resources though are not reclaimed at that time, the file is just renamed to a hidden name that includes the deletion timestamp. During the master's regular scan of the file system namespace, it removes any such hidden files.

CHAPTER 3

ALGORITHMS

3.1 Approach with DisC Diversity

3.1.1 Original DisC Diversity Algorithm

Quality is extremely appreciated in the query results that are presented to the user. Result diversification is used as a means to enhance the quality of query results (e.g. [3]). An everyday scenario of result diversification could be the search of a customer over the internet in order to buy a tablet. A query result that contains various brands, different characteristics and different colours and shapes is intuitively more informative than a query result that presents homogeneous tablets with similar features. Nowadays, the need for enhancing the quality of query results is bigger due to the extensive amount of stored data. Taking into consideration the vast development of networks and the ever-growing use of applications in our life, data ought to be presented in a smarter way. Many ways of diversifying data have been proposed so far with different aspects of diversity definitions. Some of them exploit the content of the dataset, or the similarity of the query results [3]. In that way, the objects of the dataset are categorized as similar or dissimilar to each other. Another aspect of defining diversity is with novelty [4]. In that way, objects from the dataset are valuable if they present some new information that was not presented by other objects. One more view is through semantic coverage [5]. Objects from the dataset are presented only if they belong to different categories. Most previous approaches rely on assigning a diversity score to each object and then selecting either the *k* objects with the highest score for a given *k* or the objects with score larger than some threshold. In this thesis, we take under consideration a new and intuitive definition of diversity, which is called DisC Diversity [6]. DisC abbreviation stands for diversity based on dissimilarity and coverage. The diversity with DisC is defined as follows.

Definition 3.1 (r-DisC Diverse Subset). Let *P* be a set of objects. We consider two objects *p1* and *p2* of *P* to be similar, if *dist(p1, p2)* $\leq$ *r*. The function *dist* is a distance function and the number *r* is a tuning parameter that we call radius. Given *P*, we select a representative subset *S* $\subseteq$ *P* to be presented to the user such that all objects in *P* are similar with at least one object in *S* and no two objects in *S* are similar with each other.

Thus, it is ensured that all objects in *P* are represented, or else covered, by at least one object in the resultset *S* and the selected objects of *P* that are presented to the user are dissimilar. The set *S* is called *r-Dissimilar and Covering subset* or else *r-DisC diverse subset*. The role of the tuning parameter is to strengthen or loosen the similarity calculations between objects by increasing or decreasing it. An increased radius, leads to fewer and less similar to each other resulting objects. On the contrary, a decreased radius leads to larger and less diverse subsets. These operations are called zooming-out and zooming-in respectively.

One example that depicts the functionality of the DisC Diversity is the following. Suppose that a query result, that is made by a traveller who aims to travel around Europe, contains all the major european cities. The traveller cannot visit all the cities, so a representative subset should be selected. A radius *r* is defined as the tuning parameter and euclidean distance is used as the distance function, since the example contains two-dimensional data. Figure 3.1.a shows an r-DisC diverse subset that consists the answer to the traveller's needs, given the radius *r*. The cities that are highlighted in Figure 3.1.a are representing all the major cities of Europe, while not being similar to each other. The tuning parameter can be increased or decreased in order to obtain less or more cities in the final resultset, which are less similar or less dissimilar to each other respectively. In Figures 3.1.b and 3.1.c, the above two alternatives are illustrated. Finally, the extra feature of the DisC Diversity is depicted in Figure 3.1.d, which is called local zooming-in or local zooming-out. In this example, the traveller can obtain a new diverse subset with local zooming-in given a specific result.

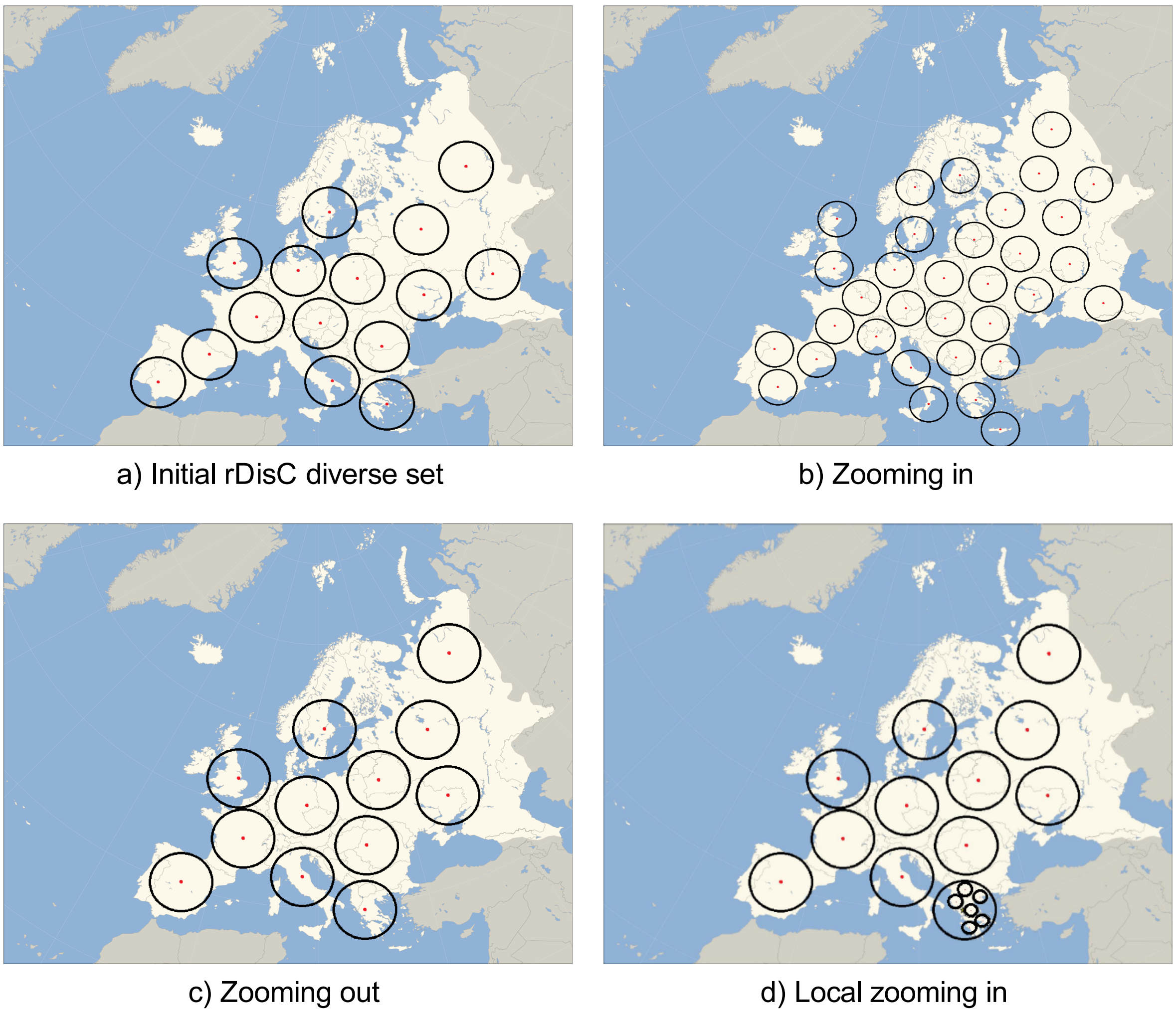


Figure 3.1: Examples of r-DisC Diverse Subsets

According to [6], the baseline algorithm for obtaining a diverse subset with DisC is the following. Let us call black the objects of *P* that are contained in *S*, grey the objects of *P* that are covered by objects in *S* and white the objects of *P* that are neither black nor grey. Initially, the set *S* is empty and all objects of *P* are white. The baseline algorithm proceeds as follows: until there are no more white objects in *P*, it selects an arbitrary white object p_i, colors p_i black and colors all objects in the neighborhood of p_i grey. The term $N_r(p_i)$ is used to denote all the neighbors of an object p_i, or else all the similar objects, given the euclidean distance metric and the radius *r*. Respectively, the term $N_r^w(p_i)$ is used to denote only the white neighbors of an object p_i.

3.1.2 DisC Diversity Parallel Variations

In our study we implement three parallel variations of the DisC Diversity algorithm. The implementation is made following the Map-Reduce logic, so the algorithms can be executed in a distributed environment. The parallel variations of the DisC Diversity are divided in two main parts, i.e. the Map phase and the Reduce phase. During the Map phase of all variations the objects of the initial dataset are split in partitions in a pre-defined manner. The partitioning of the initial dataset is a step prior to the actual algorithm and is extensively explained in 3.3. The difference of the DisC variations is located in the Reduce phase, where the actual logic of the algorithm is applied. In this step the objects from *P* are selected to participate in the resulting subset *S* with a specific logic. Every object that is selected is coloured black and the objects that exist in $N_r^w(p_i)$ are coloured grey. This procedure continues until no white objects exist in the partition that the Reducer is processing. Finally, the black objects that constitute the r-DisC diverse subset *S* are presented to the user. The objects that are processed by DisC algorithm can be multi-dimensional and the distance function could be anything (e.g. manhattan distance metric). In our implementation, the distance function that is used for the DisC variations is the euclidean distance, since the datasets that are processed are composed of two-dimensional objects. The calibration of the radius *r* remains the same as in the original DisC study. More specifically, the radius is set to 0.05, which resembles a ~1% similarity based on the dataset magnitude.

Basic-DisC Algorithm

This basic algorithm consists the first approach of this study. The Basic-DisC variation resembles the baseline algorithm of [6]. In every step, an arbitrary object from *P* is selected to be included to the resulting subset *S*.

Algorithm 1 Basic DisC Algorithm

```
Require: set P, radius r
Ensure: An r-DisC diverse subset S
 1: $S = \emptyset$
 2: for all $p_i \in P$ do
 3:     color $p_i$ as white
 4: end for
 5: while $P$ contains white objects do
 6:     select arbitrary white $p_i$
 7:     $S = S \cup \{p_i\}$
 8:     color $p_i$ as black
 9:     for all $p_j \in N_r^w(p_i)$ do
10:         color $p_j$ as grey
11:     end for
12: end while
13: Return $S$
```

Center-DisC Algorithm

An issue that comes up with the random selection of Basic-DisC is the inclusion of similar objects in the resulting subset. This situation is faced when objects, that exist near the cutting edges of the partitions, are selected to be part of the resulting subset. Center-DisC algorithm intends to solve this problem. The idea is to choose every time white objects to include to the set *S*, that are closer to the center of the partition.

Algorithm 2 Center DisC Algorithm

```
Require: set P, radius r
Ensure: An r-DisC diverse subset S
 1: $S = \emptyset$
 2: for all $p_i \in P$ do
 3:     color $p_i$ as white
 4: end for
 5: while $P$ contains white objects do
 6:     select most central white $p_i$
 7:     $S = S \cup \{p_i\}$
 8:     color $p_i$ as black
 9:     for all $p_j \in N_r^w(p_i)$ do
10:         color $p_j$ as grey
11:     end for
12: end while
13: Return $S$
```

Greedy-DisC Algorithm

This variation resembles the original greedy DisC algorithm. The flavor of this variation is that every time the object with the largest neighborhood of white, else uncovered, objects is chosen to be included in the resulting subset *S*. Thus, the coverage of the object that is selected is maximized and the resulting subset is minimized. The target is to select as few objects as possible to present to the user, which cover at the same time the whole initial dataset.

Algorithm 3 Greedy DisC Algorithm

```
Require: set P, radius r
Ensure: An r-DisC diverse subset S
 1: $S = \emptyset$
 2: for all $p_i \in P$ do
 3:     color $p_i$ as white
 4: end for
 5: while $P$ contains white objects do
 6:     select white $p_i$ with largest $|N_r^w(p_i)|$
 7:     $S = S \cup \{p_i\}$
 8:     color $p_i$ as black
 9:     for all $p_j \in N_r^w(p_i)$ do
10:         color $p_j$ as grey
11:     end for
12: end while
13: Return $S$
```

It has already been mentioned that the actual calculations of the DisC variations are conducted within the Reduce phase. However, during the Map phase, the initial data are split with a specific logic. Two partitioning methods are utilized in order to accomplish this.

3.1.3 Data Partitioning

Grid Partitioning

This procedure happens right within the Map phase. The space is split in equally sized chunks and the objects of these chunks are distributed to different Reducer functions. This way of partitioning is more applicable in datasets with objects that follow a uniform distribution. An example of grid partitioning is shown in Figure 3.3, where the original space of the initial dataset is split in 9 chunks.

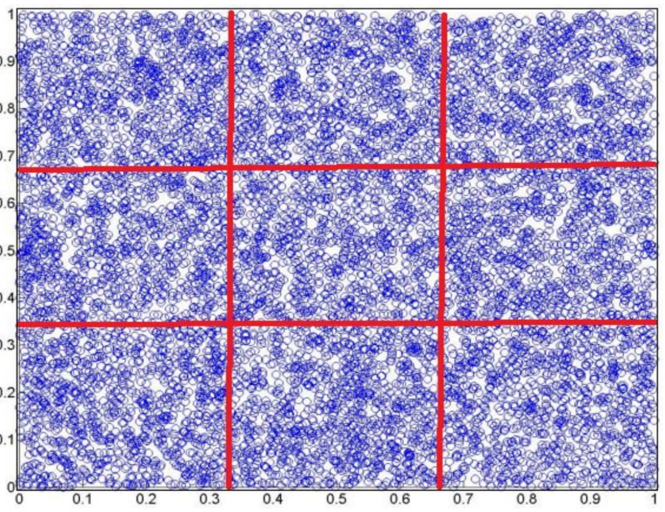


Figure 3.3: Grid Partitioning of a Uniform Dataset in 9 Parts

K-means Partitioning

The grid partitioning method is considered too simplistic, when issuing datasets with objects that follow a normal distribution. Thus, we utilize the k-means partitioning as a clustering method, which adjusts better to the structure of the dataset. In particular, a distributed variation of k-means was applied, which is called Parallel-K-Means and is extensively described in [17]. PKMeans algorithm is a parallel implementation of the common k-means algorithm. A brief explanation on the logic of the algorithm could be the following. Similar to the original k-means, a set of *n* points from the initial dataset is pre-defined randomly as the centers of the partitions. In the Map step, the distances of all the points from the selected centers are calculated and every point is assigned to the nearest center. In the Reduce step, the centers are re-estimated by applying a mean function in all the points of every partition. This procedure is repeated until the estimation of the centers in one step does not differ from the estimation in the previous step over a threshold. The whole procedure of the k-means clustering takes place as a prior step of the actual diversity algorithm and provides the DisC variations with a list of hardcoded centers of every partition. Based on these centers the Mappers perform the data splitting. Following Figure 3.4 displays the centers that were produced by the PKMeans when tested on a synthetic dataset, which follows a normal distribution. In this example the 9 obtained centers form a voronoi diagram, which visualize quite satisfactory the previously described use case.

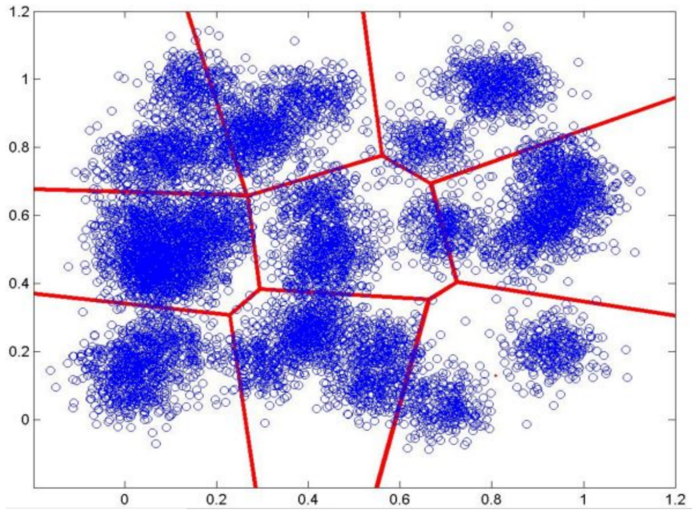


Figure 3.4: K-means Partitioning of a Normal Dataset in 9 Parts

3.2 Approach with Maximum Coverage

3.2.1 Max Cover Algorithm

Diversifying data could be aligned in a way with the maximum coverage problem. A brief explanation of this problem is as follows. Given several sets of data that could have some elements in common and a number k, at most k of these sets should be selected such that the maximum number of elements are covered, i.e. the union of the selected sets has maximal size. The maximum coverage problem needs an approximation algorithm and the predominant one has been the greedy algorithm.

As shown below, the greedy algorithm selects at each step a set which contains the largest number of uncovered elements. It can be shown that this algorithm achieves an approximation ratio of $1 - \frac{1}{e}$. Inapproximability results show that the greedy algorithm is essentially the best possible polynomial time approximation algorithm for maximum coverage problem.

Algorithm 4 The Max Cover Greedy Algorithm

```
Require: $S_1, ..., S_m$ and an integer $k$
1: while $k > 0$ do
2:     Let $S$ be a set of maximum cardinality
3:     Output $S$
4:     Remove $S$ and all elements of $S$ from other remaining sets
5:     $k = k - 1$
6: end while
```

In our study we utilized a parallel implementation of the Max Cover algorithm, that was proposed in [9] and exploits the Map-Reduce framework. The main idea behind this implementation is to simultaneously choose many sets to include in the solution, instead of selecting one at each step. This objective cannot be met at every step though and identifying when to perform a parallel choice is the crux of this algorithm. In this regard, inspiration was derived from the parallel algorithm for the set cover problem.

The MRGreedy algorithm typically requires a ground set, which is composed by the amount of the initial objects, and a set system, which is a series of overlapping sets that contain random number of objects. The output is an ordering of the sets that constitute the set system, such that for every *k* sets selected, maximum coverage of the initial objects is achieved. The algorithm approximation is slightly worse than the one of the greedy algorithm, and can be implemented to run in polynomial map-reduce steps over the input instance. Thus, it ends up nearly matching the performance of the greedy algorithm while obtaining a solution that can be implemented in the scalable Map-Reduce framework. The following algorithm, from [9], describes the functionality of MRGreedy.

The MRGreedy algorithm is typically a linear double for-loop that consists of a number of Map-Reduce steps and is executed until a condition is met. Lines 3, 5, 7, 10, 11, 13, 15 of the algorithm are performed by merging Map and Reduce operation along with the transpose operation. With the transpose operation it is able to convert tuples from <i, S_1, S_2, …, S_k>, which means that element i is contained is the following sets, to <S_1, i_1, i_2, …, i_M>, which means that set S_1 contains all the following elements.

Algorithm 5 The MrGreedy algorithm.

```
Require: A ground set $X$, a set system $\mathcal{S} \subseteq 2^X$.
 1: Let $\mathcal{C}$ be an empty list
 2: for $i = \lceil \log_{1+\epsilon^2} |X| \rceil$ downto 1 do
 3:   Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
 4:   for $j = \lceil \log_{1+\epsilon^2} \Delta \rceil$ downto 1 do
 5:     Let $X' = \{x \mid x \in X \wedge \deg_{\mathcal{S}_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
 6:     while $X' \neq \emptyset$ do
 7:       if there exists $S \in \mathcal{S}_w$ such that $|S \cap X'| \geq \frac{\epsilon^6}{1+\epsilon^2} \cdot$
          $|X'|$ then
 8:         Append $S$ to the end of $\mathcal{C}$
 9:       else
10:         Let $\mathcal{S}_p$ be a random subset of $\mathcal{S}_w$ chosen by
            including each set in $\mathcal{S}_w$ independently with
            probability $p = \frac{\epsilon}{(1+\epsilon^2)^j}$
11:         if $\left|\bigcup_{S \in \mathcal{S}_p} S\right| \geq |\mathcal{S}_p| \cdot (1+\epsilon^2)^i \cdot (1-8\epsilon^2)$ then
12:           We say that an element $x$ is bad if it is con-
              tained in more than one set of $\mathcal{S}_p$
13:           A set $S \in \mathcal{S}_p$ is bad if it contains bad ele-
              ments of total weight more than $4\epsilon \cdot (1+\epsilon^2)^i$
14:           Append all the sets of $\mathcal{S}_p$ that are not bad
              to the end of $\mathcal{C}$ in any order
15:           Append the bad sets of $\mathcal{S}_p$ to the end of $\mathcal{C}$ in
              any order
16:       Remove all the sets in $\mathcal{C}$ from $\mathcal{S}$
17:       Remove all the elements in $\bigcup_{S \in \mathcal{C}} S$ from $X$ and
          from the sets in $\mathcal{S}$
18:       Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
19:       Let $X' = \{x \mid x \in X \wedge \deg_{\mathcal{S}_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
20: Return the list $\mathcal{C}$
```

3.2.2 All r-Neighbors Algorithm

Formally, the nearest-neighbor search problem is defined as follows: given a set *S* of points in a space *M* and a query point $q \in M$, find the closest point in *S* to *q*. A direct generalization of this problem is a k-nearest-neighbor search, where we need to find the k closest points. Typically, *M* is a metric space and dissimilarity is expressed as a distance metric, which is symmetric and satisfies the triangle inequality. Most commonly, *M* is considered to be the d-dimensional vector space where dissimilarity is measured using the Euclidean distance, Manhattan distance or other distance metric. However, the dissimilarity function can be arbitrary. One example are asymmetric Bregman divergences, for which the triangle inequality does not hold. An all-k-nearest-neighbor query, i.e. an AkNN query, is a variation of a k-nearest-neighbor query. It determines the k nearest neighbors for each point in the dataset.

In [13], a new method for efficiently processing AkNN queries in Hadoop is proposed. The basic idea is to decompose the target space into smaller cells. A parameter is given in order to specify the granularity of the decomposition. At the first phase, the entire dataset is scanned and a summary of the point distribution is retrieved. According to the information of the first step, an appropriate cell decomposition is determined. In the following phases, k nearest neighbor objects for each data point are defined by considering the maximal range in which possible objects are located.

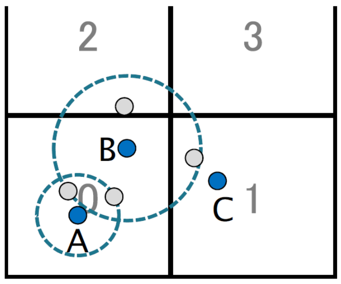


Figure 3.2: Cell Based k-NN Processing

The expected position of the k nearest neighbors is considered to be in the same cell with the object. There is a possibility though that a k nearest neighbor set includes objects from nearby cells too. To illustrate this situation, consider Figure 3.2, where an AkNN query for k=2 is processed. We can find 2-NN points for A by only investigating the inside space of cell 0. The circle centered at A, which is called a boundary circle, tightly covers the 2-NN objects and does not overlap the boundary of cell 0. In contrast, the boundary circle for B overlaps with cells 1, 2, and 3. In this case, there is a possibility that we can find 2-NN objects in these three cells. Therefore, an additional investigation is necessary, with an extra step in the algorithm.

The motivation behind the utilization of the AkNN algorithm is the need to provide the MRGreedy with a series of sets of objects that constitute the set system. These sets are going to be the neighborhoods of the objects, which are calculated by the euclidean distance metric and a radius *r*. Following the guidelines of the original work [13], we implemented the All r-Neighbors algorithm that fits our needs.

```
Algorithm 6: All r-Neighbors Algorithm
  Input: The initial objects and a radius r
  Output: List with the neighbors for every initial object
  Map Reduce 1:
    Separate the objects in cells of 2r*2r dimensions and calculate the
    neighborhood of every object inside those cells.
  Map Reduce 2:
    Associate objects with neighboring cells that could contain possible
    neighbor objects and calculate neighborhoods inside those cells too.
  Map Reduce 3:
    Integrate the lists with the neighbors of every object in order to eliminate
    duplications.
```

The algorithm consists of three basic Map-Reduce steps. The input of the algorithm is a collection of tuples of the two-dimensional points <*id, xCoord, yCoord*> and a radius *r*. The output of the algorithm is a collection of tuples that represent the *id* of a point along with its neighbors, such as <*id, [neighborList]*>. An example application of Algorithm 6 is illustrated in the appendix.

3.2.3 Combined Max Cover Algorithm

The maximum coverage approach on the data diversification problem is motivated mainly by the design of the DisC variations. More specifically, the fact that the initial dataset is partitioned and processed by different Reducers may lead to the selection of similar objects in the resulting subset, especially nearby the cutting edges of the partitions. A new algorithm intends to overcome this situation, which is called Combined Max Cover algorithm. This algorithm is a combination of the above Algorithms 5 and 6. The MRGreedy algorithm, apart from the initial points per se, requires also a collection of sets of data, among which it selects to acquire in the resulting dataset. In order to maintain the same input parameters as those of the DisC variations, i.e. the initial dataset and a radius, we utilize the clustering method of All r-Neighbors in order to provide the desired collection of sets of data to the MRGreedy.

```
Algorithm 7: Combined Max Cover Algorithm
Input: The initial objects and a radius r
Output: The objects in an order that maximum covers the initial
        dataset for any k objects chosen
Step 1:
 Utilize All r-Neighbors algorithm to produce neighborhoods for every
 object
Step 2:
 Utilize MRGreedy algorithm to produce an ordering of the neighborhoods
Step 3:
 Output the ordering of the initial objects that maximum covers the
 initial dataset for any k objects chosen
```

The input for this algorithm is the initial dataset and a radius *r*, as in the DisC variations, whereas the output is an ordering of the initial objects, such that for any *k* objects selected, the maximum number of initial objects is covered. The radius *r* still holds the role of the tuning parameter in this algorithm. The size of the perceived neighborhoods of every object during the All-r-Neighbors algorithm step is affected by this parameter. Finally, one more feature of this algorithm is the flexibility in the cardinality of the resulting dataset. Parameter *k* gives the opportunity to the user to keep a variable number of resulting objects.

3.3 Proposed Algorithms in Brief

Concluding, in this paragraph we summarize the proposed algorithms that have been used in our experiments. The two main categories of algorithms are based on DisC Diversity and Max Cover respectively. The differentiator between these categories lie in the perspective that we approach the data diversification problem. In the first case the problem is solved through a partition-based variation of the DisC algorithm implemented in Map-Reduce, whereas in the second case maximum coverage algorithm is considered as a means of diversifying data.

Concerning the first category, three variations of the DisC Diversity algorithm are utilized in this study. The difference is observed in the way that the objects are selected to be included in the final resultset, i.e. randomly, centrally or in a greedy way. The methods are applied in datasets that are partitioned in two different ways, in a pre-defined manner or with k-means respectively. The above statement is better illustrated in Table 3.1, which consists a point of reference in this study for all the possible combinations of the DisC variations and the data partitioning methods.

Table 3.1: DisC Variation Algorithms Combined with Partitioning Methods

	Grid Partitioning	K-means Partitioning
Basic-DisC	BP1	BP2
Center-DisC	CP1	CP2
Greedy-DisC	GP1	GP2

Concerning the second category, the diversification problem is associated with the maximum coverage problem. The Combined Max Cover algorithm, or else CMC, is utilized through a combination of the All r-Neighbors and the MRGreedy algorithms in order to produce an ordering of the initial objects such that any *k* objects selected ensure always the maximum coverage of the initial dataset.

CHAPTER 4

EXPERIMENTAL SETUP

4.1 Technical Specification

The goal of this study is to perform a parallel implementation of the aforementioned algorithms in order to be able to run in a distributed environment. We take advantage of the Hadoop framework, which is an open source platform, distributed by Apache. Hadoop framework offers many features, among which the Hadoop File System and the Map-Reduce library. We have installed the latest version of Hadoop, at the time when the experiments were conducted, which was hadoop-1.1.2. The formal installation procedure has been followed based on the official guide [14], along with another one [15], which was very helpful. Some necessary changes have also been made in order for the Hadoop to run properly, such as installing the suitable java v7 libraries or disabling the IPv6 communication between the machines of the cluster, that participated in the experiments. In addition, we have used java programming language within the Eclipse IDE and took advantage of the Map-Reduce plugin, which gives the opportunity to run Map-Reduce tasks locally for debugging purposes.

The experiments were conducted using the cluster of commodity computers that is provided by our department. Some technical information about the cluster is the following:

- The cluster consists of 15 nodes.

- Each node is a computer with a dual-core with HT CPU operating at 2GHz and a 4GB RAM.
- The nodes communicate through a gigabit ethernet connection.

The Hadoop File System utilizes the nodes of the cluster by giving them the role of the Master, which coordinates the overall functionality, or the Worker, which processes the actual data. Moreover, the Map-Reduce framework utilizes the nodes with a role of Mapper or Reducer, regarding the phase of the procedure. The numbers of Mappers and Reducers that participate in a Map-Reduce procedure is configurable. Our experiments were conducted with various numbers of Mappers and Reducers, in order to investigate the impact of this variation in the algorithms results.

In a nutshell, the utilization of the Hadoop framework consists of the following. The initial data that are to be processed should be transferred inside the HDFS with appropriate hadoop commands, which are illustrated in the appendix. A jar file should be created, which contains the Map-Reduce program, i.e mapper, reducer and driver classes. The command that runs a jar file inside the HDFS should be utilized, in order to produce the outcome of the algorithm. Finally, the output files are retrieved from the HDFS with respective commands.

4.2 Datasets

The first category of datasets that are used in order to evaluate the proposed algorithms are synthetic. These datasets consist of two-dimensional points inside the geometric space of [0,1]x[0,1] and represent a resultset of a query in a database. The two dimensions simulate the characteristics of every object in the resultset. Consider for example a resultset of a query which returned cameras as objects. The characteristics could be the brand-name of the camera, the resolution of the lens, the number of colours that the camera supports, etc. The category of the synthetic data consists of two different kinds of datasets. The first dataset is composed by objects that are created with a uniform distribution along the geometric space of [0,1]x[0,1]. This dataset was created as a first simple approach in order to validate the algorithms.

However, it emerged as valuable, since it was observed that sometimes large scale datasets tend to adopt a random behavior that could be resembled by a uniform distribution. The next dataset of this category is composed of objects that are created with a normal distribution. The logic behind the creation of this dataset is the following. We selected a random number of fixed objects with random positions within the space of [0,1]x[0,1]. Around those fixed objects we built the rest of the objects with a gaussian distribution and a random standard deviation. This kind of dataset is utilized in order to simulate the behavior of the algorithm regarding large scale clustered datasets.

Secondly, the proposed algorithms are evaluated against a real dataset as well. Following the idea of the original work [6], where DisC was firstly described, we utilize a location-based real dataset. It is originated from MaxMind [18], which is is an industry leading provider of IP intelligence and online fraud detection tools, and contains information about cities around the world. More specifically, this dataset holds information over the name of the city, the respective region, the respective country, the latitude and longitude along with the population of the city for over three million cities all over the world. We utilize the two dimensions of longitude and latitude and we adapt the parameters of the algorithms in order to provide comparable results with those of the synthetic datasets.

4.3 Evaluation Strategy

4.3.1 Definition of Error Points

An evaluation criterion that is used is the number of the error points that are contained in the resultsets of the algorithms. Normally, the output of the algorithms is supposed to contain objects that are not similar to each other, but this is not always the case. It is possible that the resultset contains objects that abstain less than the predefined radius. Such results are called *error points*. The notion of an erroneous resulting object has emerged when the DisC variations were evaluated. Prior to the actual execution of the algorithm a partitioning takes place, separating the initial data in *N* parts, where *N* was equal to the number of the reducers that participated in the experiment. The handling of different chunks of data from different reducers could lead to a resultset that contains error points.

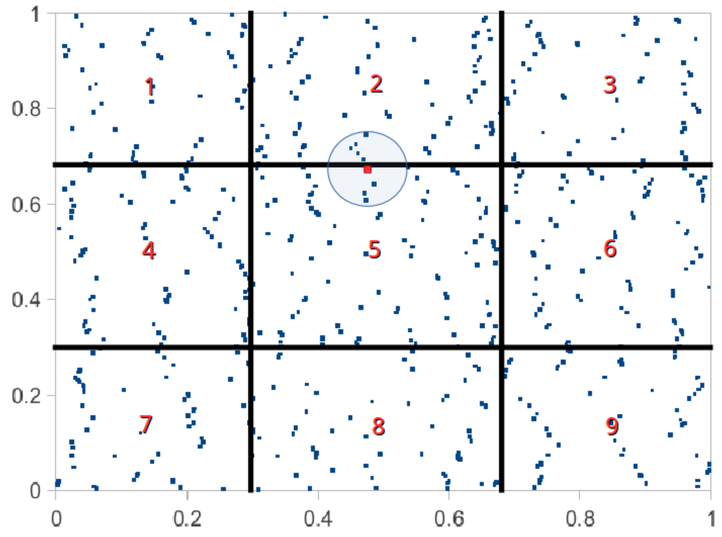


Figure 4.1: Error Point Illustration

In Figure 4.1, we can observe an example of a possible error point. The highlighted object exists physically in area 5 and it is closer than the amount of radius to the cutting edge of the partitions. As a consequence its neighborhood regarding the initial dataset consists of objects that exist in area 5 and area 2 as well. Typically, if the resulting dataset of the algorithms consists of two such objects that come from different and neighboring parts and happen to be similar, then we can safely assume that one of these objects is erroneous. Generally, error point production is the main drawback of the parallel implementation of DisC Diversity algorithm and is not expected to affect the correctness of the Combined Max Cover algorithm.

4.3.2 Evaluation of Algorithms

First step of the experimental evaluation is the analysis of the results that all the proposed algorithms produce when tested under the same testbase. The conditions under which the aforementioned experiments were held are summed up in Table 4.1, where the default values of the experiment parameters are illustrated. In more details, the experiments were conducted with the use of an initial dataset with cardinality equal to 100K objects, which were distributed in a uniform manner. Moreover, the number of mappers and reducers that participated in the experiments were both set to 15 and the radius, that regulates the similarity between two objects, has been considered equal to 0.05.

Table 4.1: Default Parameters of Algorithms Evaluation

Parameters	Initial Dataset Cardinality	Initial Dataset Structure	Number of mappers and reducers	Radius 'r'
Default values	100K objects	Uniform Distribution	15 mappers / 15 reducers	0.05

The first evaluation criterion in our experiments is the resulting dataset cardinality. More specifically, we are interested in the percentage of the resulting dataset cardinality over the initial dataset cardinality. Furthermore, apart from a resulting dataset with small cardinality, the goal is a clean and independent dataset as well. Based on the above, an additional criterion is the percentage of the error points that are contained in the resulting dataset over the total cardinality of it. Last but not least, we care about the execution time that each algorithm needs in order to function. It is a fact that Map-Reduce is commonly used for post-processing large scale datasets and execution time is not what we care about most, nonetheless in our experiments runtime measurement is needed and consists an evaluation criterion for the algorithms.

The aim of this analysis is firstly to distinguish which algorithm displays the most correct results among the DisC variations. Moreover, this analysis provides an initial comparison between the two categories of algorithms, i.e. the DisC variations and the Combined Max Cover algorithm. In order to perform this comparison we need to make an assumption, since the CMC algorithm does not stop its execution until it provides a whole ordering of the objects of the initial dataset. Thus, we consider that *k*, which is an output parameter of the CMC algorithm, is equal to the cardinality of the resultset that the most efficient DisC variation has produced. Based on this assumption, a comparison can be made regarding the error point production of each algorithm.

4.3.3 Comparison of Algorithms

After evaluating all the proposed algorithms under a common environment, a stress test is performed regarding the impact of the experiment parameters in the efficiency of the two categories of algorithms, i.e. the DisC Diversity and the CMC algorithm. Among the DisC variations, the GP1 has been chosen, as it produces the resulting dataset with the smallest cardinality and a rather low percentage of error points at the same time. Moreover, we have concluded that the extra effort for a k-means partitioning before running the actual algorithm does not provide any extra value, whilst it increases the overall execution time.

The evaluation criteria of the resultset cardinality, the error point percentage and the runtime remain the same in this analysis as well. In this evaluation though the difference of the two algorithm categories is investigated, taken that the experiment parameters are changing. In Table 4.2, the different values of the experiment parameters are illustrated.

Table 4.2: Experiment Parameters Values

Parameters	Initial Dataset Cardinality	Initial Dataset Structure	Number of mappers and reducers	Radius ‘r’
Values	10K objects	Synthetic Dataset with Uniform Distribution	9 mappers / 9 reducers	0.025
	100K objects	Synthetic Dataset with Normal Distribution	18 mappers / 18 reducers	0.05
	1M objects	Real Dataset	27 mappers / 27 reducers	0.1
			36 mappers / 36 reducers	

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Proposed Algorithms Evaluation

An evaluation of all the algorithms under investigation follows in this paragraph, based on the evaluation criteria that we have set, which are the cardinality of the resulting subset along with the error points percentage and the execution time of each algorithm. The testbase in the evaluation of all the algorithms remains the same. The experimental setup is illustrated in Table 4.1, where the default parameters are shown. More specifically, the experiments are performed on a dataset with a cardinality of 100K objects that are uniformly distributed. Moreover, the radius is set to 0.05 and 15 Mappers and Reducers respectively are contributing simultaneously.

The Greedy-DisC algorithm, which proceeds by choosing objects with the maximum uncovered neighborhood to include in the resultset, outputs the r-DisC diverse subset with the smallest cardinality. In this algorithm, the coverage of every chosen object is maximized and thus the cardinality of the resulting subset is minimized. Moreover, the error points that are produced with Greedy-Disc hold the smallest percentage, which makes the resulting subset of this algorithm the most accurate. The Center-DisC algorithm, produces the resultset with the biggest cardinality. This happens mainly because of the nature of this algorithm, which is to select objects that exist nearest to the center of the partition and avoid the selection of possible error points that exist nearby the cutting edges. This behavior leads to the selection of objects that are closer to the ones that were previously selected. The outcome of the error points percentage of Center-DisC, under the specific testbase, does not depict exactly the behavior of the algorithm. In paragraph 3.3, more test results with different datasets are illustrated in order to describe the functionality of Center-DisC. Last but not least, the Basic-DisC algorithm leads to a resultset with cardinality among the other two and an increased number of error points. The randomness of this algorithm may not define it as the most efficient one, but it remains valuable, since simplicity is what matters in Big Data processing.

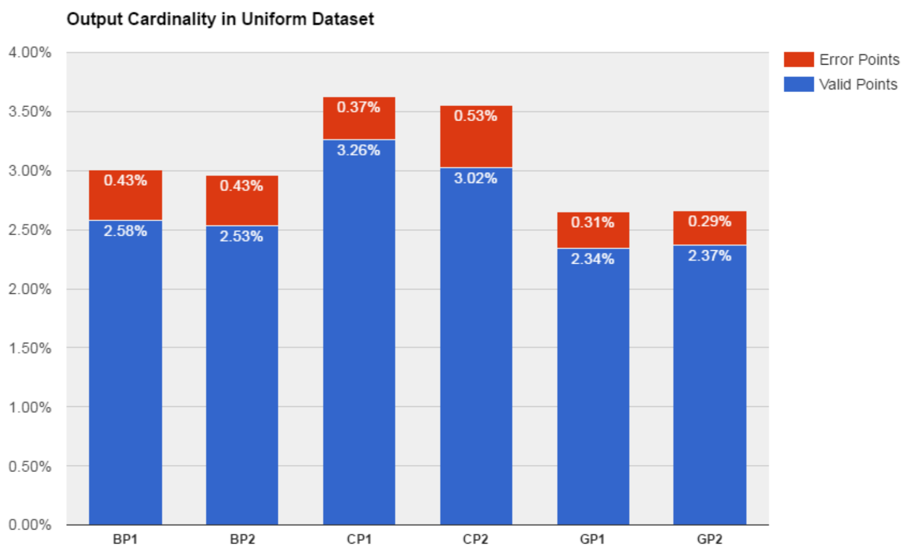


Figure 5.1: Output Cardinality for different DisC Variations and Data Partitionings in Dataset with Uniform Distribution

The CMC algorithm provides an ordering of the objects of the initial dataset, rather than a resultset. Following is a comparison between GP1, which is the most efficient among the DisC variations, and CMC based on the error points percentage. Due to the fact that the algorithms do not share the same input parameters, the comparison is made by considering the output parameter *k* of CMC to be equal to the cardinality of the resulting subset of GP1. Furthermore, the same radius, as in GP1, is used in order to define the neighborhoods in the A-kNN step. GP1 seems to produce more error points that the CMC algorithm. The main reason is that GP1, along with all the DisC variations, is a partitioning based method. The main drawback of this method consist the cutting edges that exist between the partitions. In areas close to the cutting edges, there can exist objects of the resulting subset that are normally similar in the initial dataset, which makes them erroneous and thus the resultset gets more inaccurate.

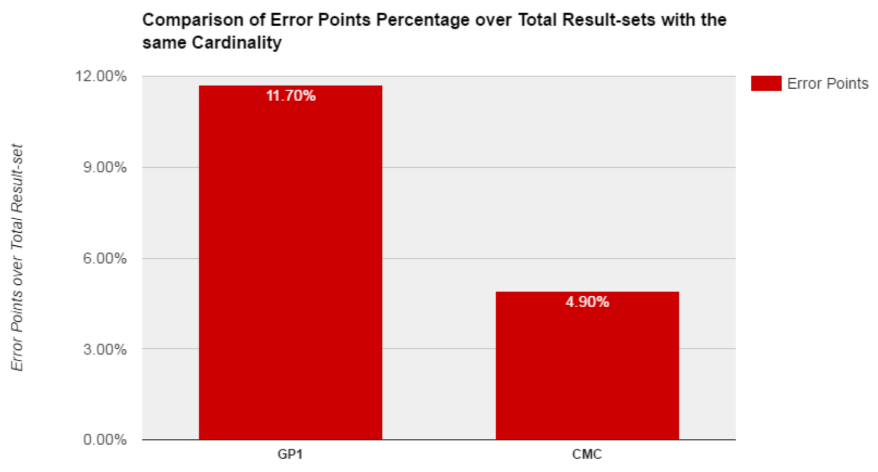


Figure 5.2: Error Points Comparison for GP1 and CMC

Below, in Figure 5.3, the execution times of all the proposed algorithms are illustrated regarding the experiments that were made with the given parameters of Table 4.1.

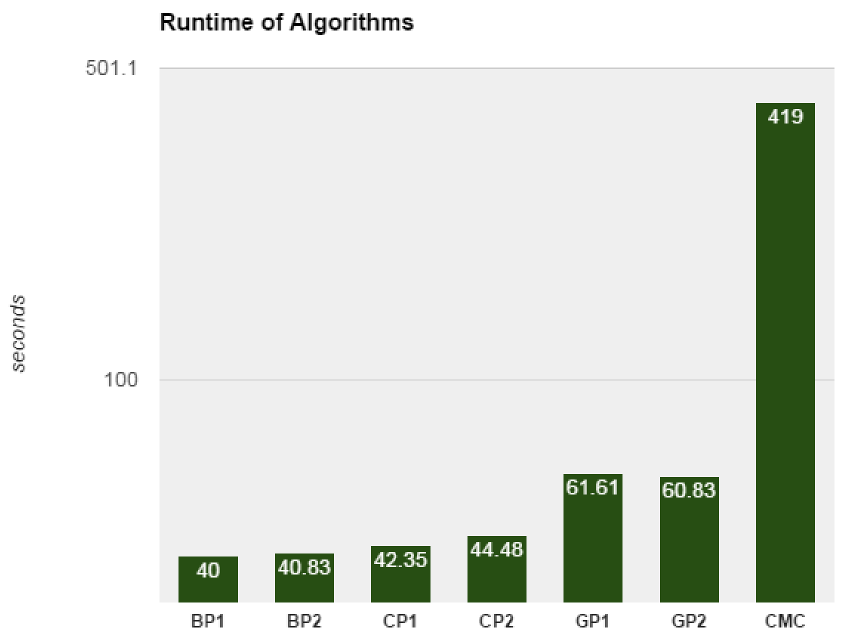


Figure 5.3: Execution Times of the Derived Algorithms

As we can observe in Figure 5.3, CMC indeed has a higher execution time than the DisC variations. This is due to the loops that are needed in order to select sets to include to the resultset and the initial startup cost. The importance though for our experiments is not the actual execution time but the rate that it increases in comparison to the different input parameters. Among the DisC variations, Basic-Disc is the fastest one and Greedy-Disc, which needs more computational cost in order to apply its logic in the Reduce phase, is the slowest.

Regarding the DisC variations, the consumed time is not actually evenly distributed. It is a fact that the parallel implementation of the DisC Diversity algorithm needs more computational cost, and hence runtime, within the reduce phase. The map phase does not seem to add an important overhead on the runtime of the algorithm. On the contrary, the pre-execution phase of k-means partitioning adds a worth mentioning amount of time in the total runtime of the DisC variations, mainly due to the initialization setup and the level of accuracy that is needed. Regarding the CMC algorithm, most of the time seems to be consumed in the phase of the MRGreedy execution. Even if many of the steps of the algorithm have been parallelized in order to be able to be executed in Map Reduce, the core of the algorithm runs in a sequential manner until a condition is met. The algorithm is able to be executed over polynomial map reduce steps over the input size but the actual execution time depends also on the structure of the initial dataset. The neighborhoods of the objects are relatively fast calculated with the All r-Neighbors algorithm.

5.2 Partitioning Methods Evaluation

In this paragraph, we present the value of the different ways of partitioning of the initial data. The two ways that are used are the grid partitioning and the k-means partitioning, which are implemented prior to the actual execution of the DisC variations. In Center-DisC, regardless if the distribution of the initial dataset was uniform or normal, it is observed that k-means partitioning helps the algorithm to produce resulting subsets with smallest cardinality, in contrast to the grid partitioning. In the following Table 5.1, we illustrate the impact of the k-means partitioning method on the error points production in contrast to the grid partitioning, regarding every DisC variation and data distribution. The percentage that is shown in the table is the difference that is observed in error points percentage over the total resulting subset between the k-means and grid partitioning utilization.

Table 5.1: Impact of K-means Partitioning in Error Point Production in contrast to Grid Partitioning

		Data Distribution	
		Uniform	Normal
DisC Algorithm Variation	Basic	+ 0.24%	- 1.46%
	Center	+ 4.74%	- 2.69%
	Greedy	- 0.80%	- 1.81%

The most valuable result is seen in Center-DisC, when tested against a dataset with normal distribution. There, we observe a reduction in error points up to 2.69%. Greedy-DisC seems to enjoy a reduction in error points, regardless the initial data distribution. Nevertheless, the outcome of this experiment is not so fruitful, since the difference in error points production is ambiguous for different types of algorithms or dataset structures. We do not notice great improvements in the outcome of the algorithms and at the same time the overall runtime of the algorithms is increased, because the parallel k-means algorithm has to be executed first. Tests proved that producing a clustered dataset, prior to the actual DisC algorithm execution, increases the overall runtime up to 5 times. This is mainly because k-means algorithm depends on the initialization setup and needs a lot of iterations. As a consequence, the utilization of k-means partitioning is considered to offer little added value in the proposed algorithms, and thus it is not suggested.

5.3 Comparison of Algorithms

Finally, a comparison between the two main algorithm categories is performed, i.e the DisC Diversity and the Max Cover. In this paragraph, a comparison is conducted between GP1 and CMC in order to evaluate them, based on the criteria that we have set. More specifically, the comparison is made regarding the error points that are produced by each algorithm and their execution time. This test evaluates the algorithms from four different aspects, by changing each time one experiment parameter. The experiment parameter are shortly the initial dataset size and structure, the number of Mappers and Reducers and the radius *r*.

5.3.1 Impact of Dataset Size

The scope of this experiment is to identify the behavior of the two algorithms for input datasets with different sizes. Comparison is made on the error points that are produced from each algorithm and on their runtime as well. Table 5.2 describes the parameters of this experiment and the changing factor of initial dataset cardinality.

Table 5.2: Parameter Values for Initial Dataset Cardinality Impact Experiment

Parameters	Initial Dataset Cardinality	Initial Dataset Structure	Number of mappers and reducers	Radius ‘r’
Values	**10K objects**	Uniform	15 M/R	0.05
	100K objects	Uniform	15 M/R	0.05
	1M objects	Uniform	15 M/R	0.05

As we can verify from Figure 5.4, the error points production is the main drawback of the DisC variations. For all input sizes, GP1 maintains a higher level of error points in contrast to the CMC algorithm. Moreover, as the input dataset size get larger the rate that the error points increase gets higher as well. On the contrary, CMC seems to tackle this issue in a better way, maintaining a lower increase rate of error points production.

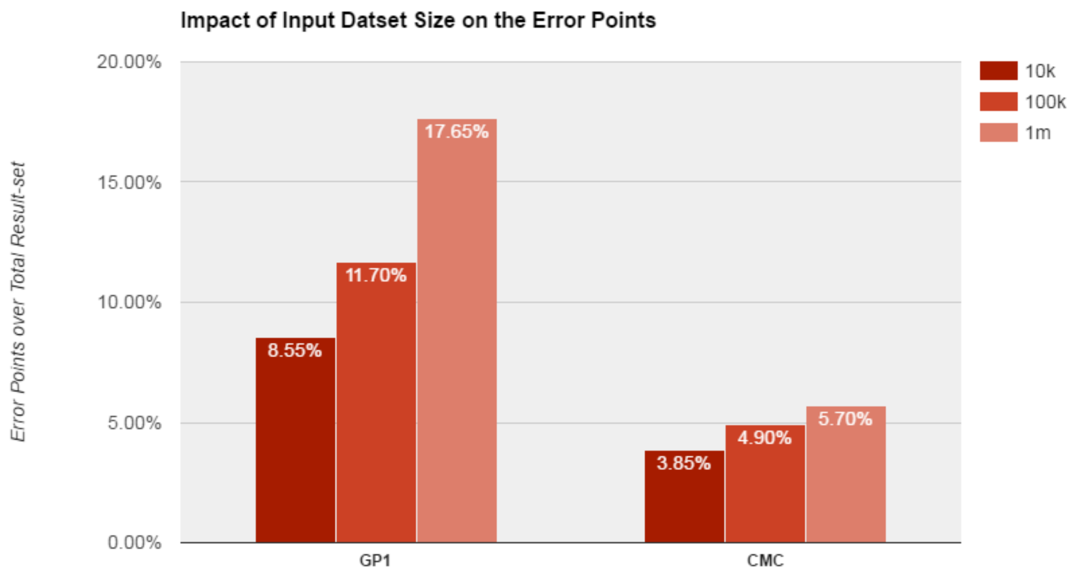


Figure 5.4: Impact of Initial Dataset Cardinality on Error Points Production

Regarding the execution time of the algorithms, the increase of the cardinality of the initial dataset seems to affect GP1 the most. In Figure 5.5, we can observe a higher increase in runtime of GP1 in comparison to the CMC algorithm. However, GP1 maintains a lower execution time for all the tested input sizes.

Nevertheless, experiments with larger datasets have shown that GP1 does not perform properly. For datasets with size of a few million objects, the DisC variations tend to stall and consume all the resources of the cluster, thus making the execution time significantly larger. On the contrary, for the same input datasets CMC tends to sustain a rather slow rate of runtime increase comparing to the increase of the input size. Figure 5.6 illustrates the increase of the execution time of CMC algorithm for different input dataset sizes.

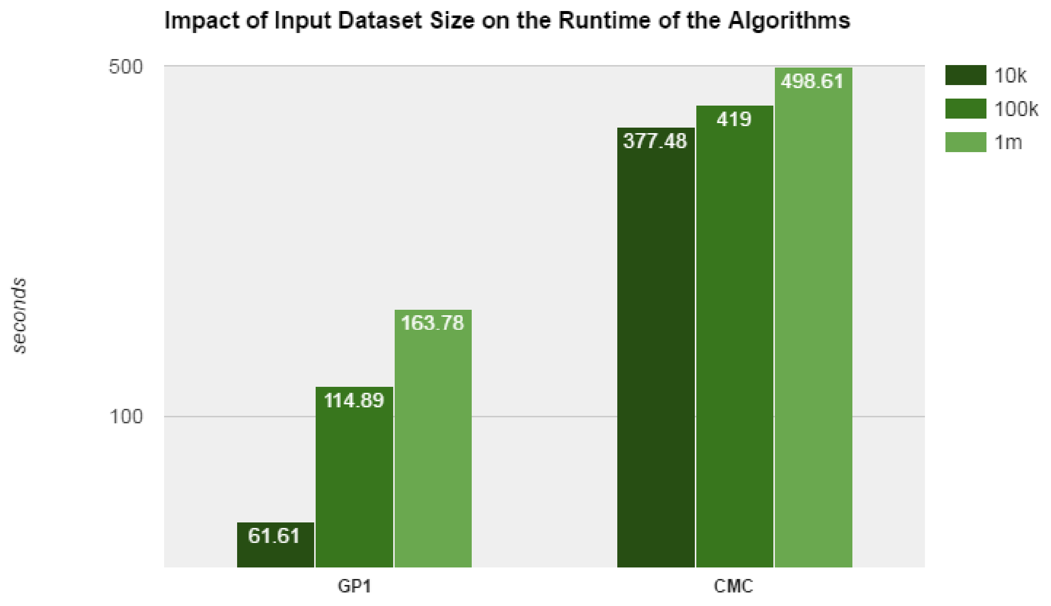


Figure 5.5: Impact of Initial Dataset Cardinality on Runtime

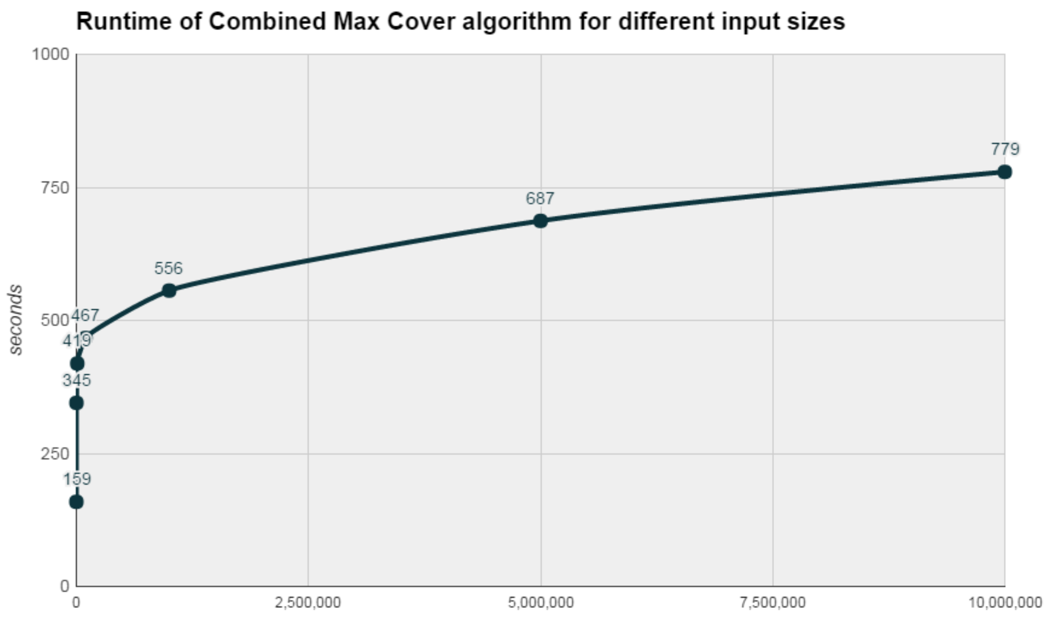


Figure 5.6: CMC Runtime for different initial dataset cardinalities

5.3.2 Impact of Dataset Structure

The target of this experiment is to identify if the structure and nature of the initial dataset affects the performance of the algorithms. GP1 and CMC are tested, representing the two algorithm categories of this study. The initial dataset structure changes, while the rest of the testbase remains the same in all three experiments. The details of the experimental environment are illustrated in Table 5.3.

Table 5.3: Parameter Values for Initial Dataset Structure Impact Experiment

Parameters	Initial Dataset Cardinality	Initial Dataset Structure	Number of mappers and reducers	Radius 'r'
Values	100K objects	**Uniform**	15 M/R	0.05
	100K objects	**Normal**	15 M/R	0.05
	100K objects	**Real**	15 M/R	0.05

The error points that are produced by the algorithms over the total resulting dataset seem to be less in a normal dataset in comparison to a uniform one. This is mainly because of the structure of the data. It was observed that clustered data lead to fewer error points, especially when processed by Greedy-DisC. The percentage of error points for a real dataset in GP1 is higher than the one of uniform or normal, but the total resulting set was significantly lower (~54%). The bigger number of error points percentage can be explained by the dataset structure, which consists of worldwide city coordinates, located in land, rather than in water. Thus the real dataset happens to be even more clustered than the synthetic one with the normal distribution. CMC algorithm has a similar behavior for all kinds of dataset structures, as we can see in Figure 5.7. The execution times of the algorithms for different dataset structures do not have significant differences. As we can see in Figure 5.8, GP1 remains the fastest algorithm, even when initial dataset structures change.

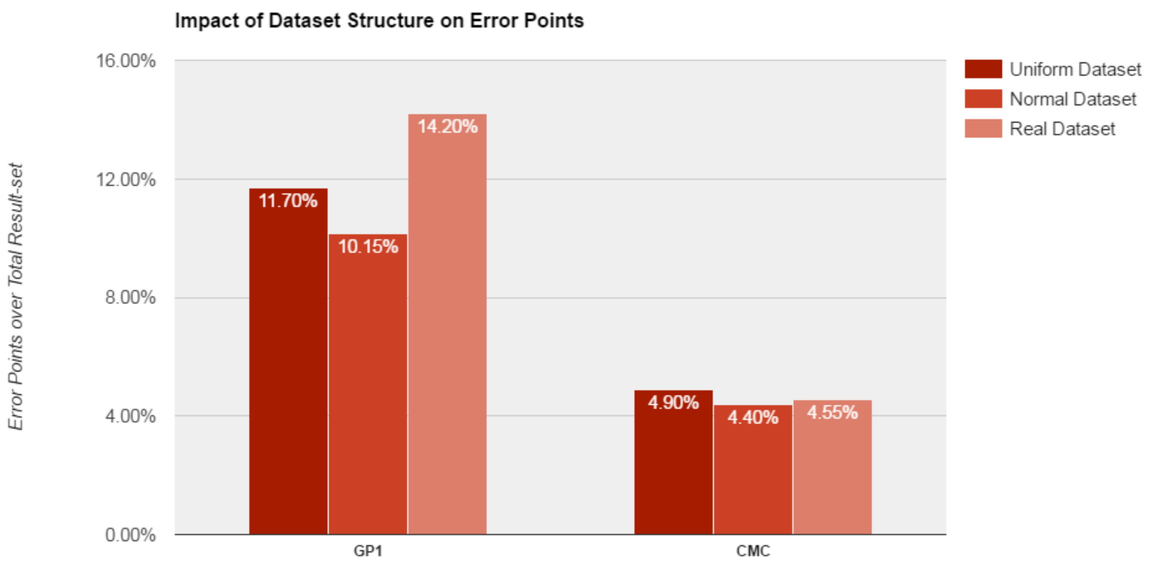


Figure 5.7: Impact of Initial Dataset Structure on Error Points Production

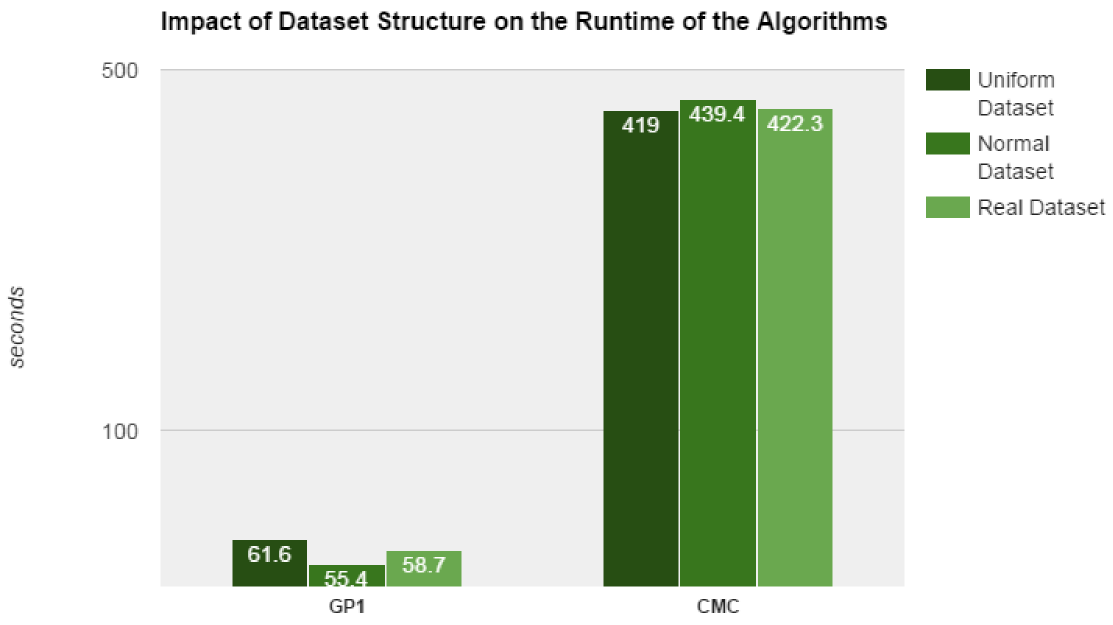


Figure 5.8: Impact of Initial Dataset Structure on Runtime

5.3.3 Impact of Number of Mappers and Reducers

Through the next experiment we intend to explore how the variation of the number of Mappers and Reducers, that participate in the processing of the data, affect the output of the algorithms. The evaluation criteria remain the same and the experiment parameters are shown in Table 5.4.

Table 5.4: Parameter Values for Number of Mappers and Reducers Impact Experiment

Parameters	Initial Dataset Cardinality	Initial Dataset Structure	Number of mappers and reducers	Radius 'r'
Values	100K objects	Uniform	**9 M/R**	0.05
	100K objects	Uniform	**18 M/R**	0.05
	100K objects	Uniform	**27 M/R**	0.05
	100K objects	Uniform	**36 M/R**	0.05

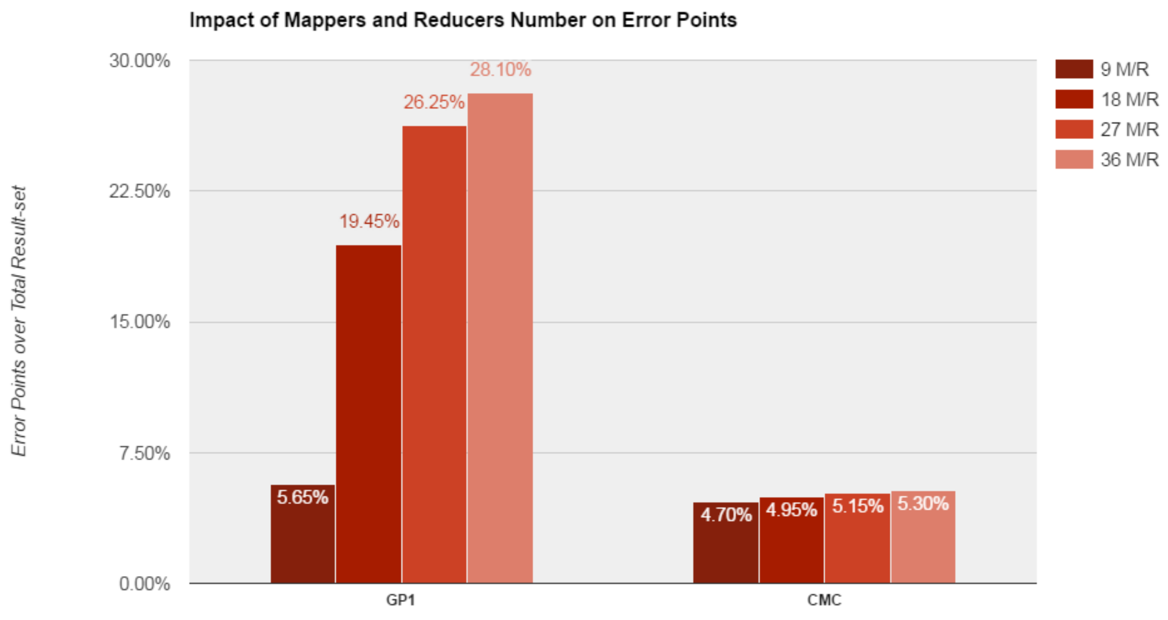


Figure 5.9: Impact of Number of Mappers and Reducers on Error Points Production

As we can observe in Figure 5.9, CMC algorithm tends to produce less error points than GP1, because it was originally designed to function in Map-Reduce and does not rely in a partitioning method. Also, the increase of the number in Mappers and Reducers does not seem to affect its results. On the contrary, the radical increase of the error points of GP1, as the number of Mappers and Reducers gets bigger, is due to the areas of the cutting edges between the partitions. Since GP1 is based in data splitting, it is extremely affected by the partitioning that is performed before the data diversification. In this experiment the number of Mappers and Reducers is bonded to the number of the partitions, affecting this way the error points production.

Figure 5.10 illustrates the different execution times of GP1 and CMC algorithms. Apart from the obvious observation that DisC has a lower runtime, we notice that the more Mappers and Reducers that are used, the less execution time is needed for both algorithms. Although, this does not hold forever, as for very high number of Mappers and Reducers that simultaneously process the algorithm, runtimes do not seem to have some added value. This is mainly because the resources are consumed more in communicational costs, rather than in the actual processing of the data.

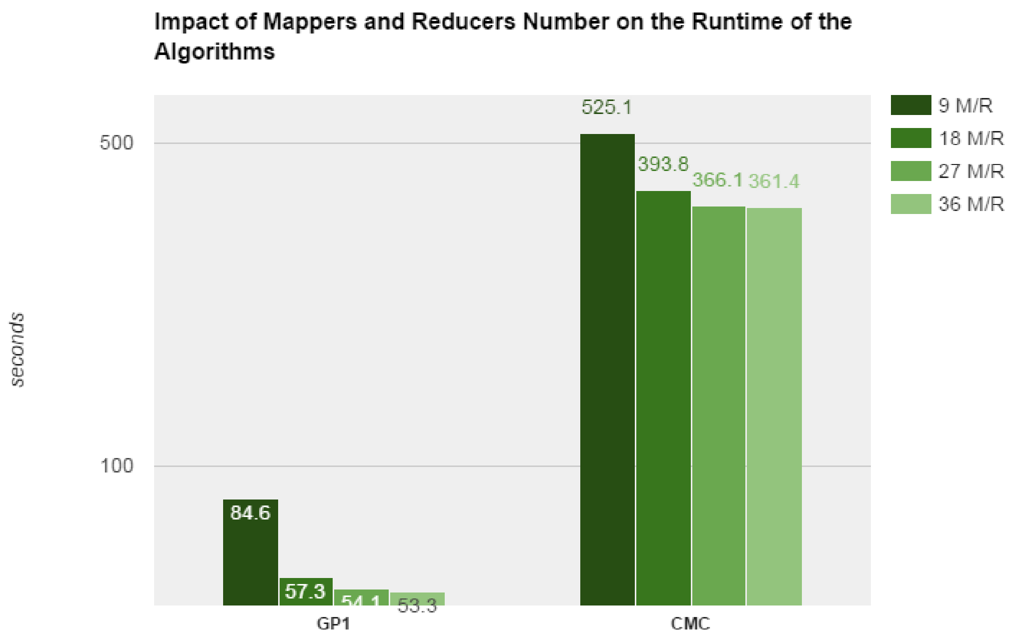


Figure 5.10: Impact of Number of Mappers and Reducers on Runtime

5.3.4 Impact of Radius

Last but not least, the experiment with the regulation of the parameter of radius is made in order to identify if radius, which defines the similarity and forms the neighborhoods of the objects, has any impact on the results of the algorithms. The evaluation criteria are once again the error points production and the runtime of the algorithms and the experiment parameters are shown in Table 5.5.

Table 5.5: Parameter Values for Radius Impact Experiment

Parameters	Initial Dataset Cardinality	Initial Dataset Structure	Number of mappers and reducers	Radius 'r'
Values	100K objects	Uniform	15 M/R	**0.025**
	100K objects	Uniform	15 M/R	**0.05**
	100K objects	Uniform	15 M/R	**0.1**

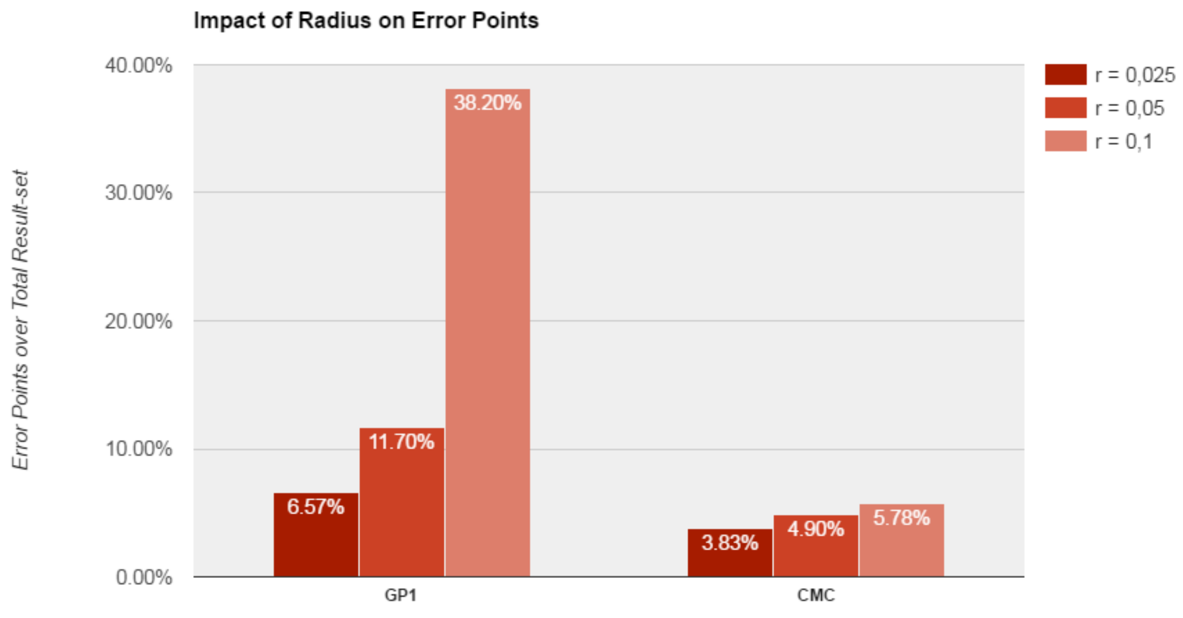


Figure 5.11: Impact of Radius on Error Points Production

As we can see in Figure 5.11, the parameter of radius seems to highly affect the error points production of GP1, whereas it has a rather minimum effect in CMC. As the radius gets larger in GP1, the objects in the resulting dataset that are closer to the cutting edges have higher possibilities to be similar with other objects from neighboring partitions and thus to be considered as erroneous. Apart from that, as Figure 5.12 illustrates, no significant differences are observed in the execution times of the algorithms as the radius parameter increases, while GP1 still remains the fastest algorithm.

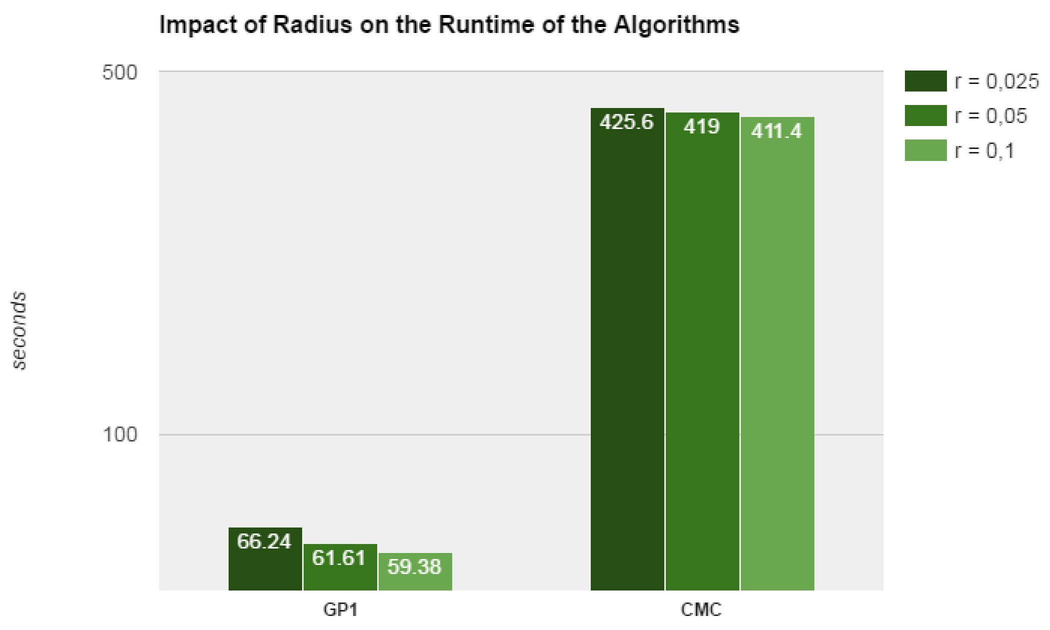


Figure 5.12: Impact of Radius on Runtime

CHAPTER 6

CONCLUSIONS

Quality has always been extremely appreciated in the query results that are presented to the user. Result diversification is used as a means of enhancing the quality of the query results. The burst of stored information nowadays along with the need for accurate query results require a parallel implementation of traditional diversification algorithms, that will be able to run in distributed environments. In this thesis, two ways of diversifying data are taken under consideration. Firstly, the DisC Diversity is studied, by implementing parallel variations of the algorithm. Then, a parallel implementation of the Max Cover algorithm in combination to the All r-Neighbors algorithm is evaluated. Both implementations follow the Map-Reduce logic. The Apache Hadoop distributed platform is used for the experiments, that were conducted in the cluster of our department.

Regarding the DisC Diversity algorithm, we made three variations that differ in the resulting objects selection, which takes place in the Reduce phase. The selection of resulting objects is made either of random objects (Basic-DisC) or objects that are closer to the partition center (Center-DisC) or objects that have the most uncovered neighborhood (Greedy-DisC). The most efficient algorithm regarding the cardinality and the accuracy of the resultset is the Greedy-DisC. It outputs the smallest r-DisC diverse subset, while maintaining the levels of error points low and close to the ones of the Center-DisC. The Center-DisC algorithm proves that a centric object selection eliminates the error points, that tend to appear in the cutting edges of the partitions. However, the cardinality of the r-DisC diverse subset that it produces is the largest one, since the objects that are selected for the resultset have a low coverage. Finally Basic-DisC outputs resultsets of medium cardinality with increased error point levels. In regards to the runtime though, basically due to the complexity of each algorithm, the results are exactly the opposite. Basic-DisC, which has no logic in the object selection, is the fastest one and Greedy-DisC, which searches among all the objects for the one with the most uncovered neighborhood, has the largest execution time.

The implementation of the DisC Diversity algorithm encapsulates a step of partitioning of the initial dataset, prior to the actual run of the algorithm. The objects of the initial dataset are partitioned either firmly in equal chunks or in shifting chunks with the use of k-means. The second way of partitioning is useful especially when the initial dataset follows a normal distribution. Generally, this enhanced way of partitioning is used in order to prevent the algorithms from producing error points. However, the experiments have shown that the utilization of an enhanced partitioning method provides little added value to the algorithm. Although a noteworthy improvement is observed regarding the error point production, a k-means clustering of the initial dataset gives an overhead to the whole runtime of the algorithm. Tests have proven that producing a clustered dataset, prior to the actual algorithm execution, could increase up to 5 times the overall runtime. This is mainly because k-means algorithm depends on the initialization setup and needs a lot of iterations.

The experiments with the parallel implementation of the DisC Diversity algorithm proved that it is not as scalable as originally expected. First of all, as the input dataset gets larger, e.g. with cardinality of a few million objects, the algorithm tends to stall the execution and the runtime increases quickly. This is somehow expected, since the actual calculations of the algorithm take place in the Reduce phase, which constitutes a bottleneck. Moreover, the utilization of more Reducers that handle different partitions of data does not seem to benefit the execution. The experiments showed that the extensive number of cutting edges between the partitions lead to an increased amount of error points in the r-DisC diverse subset.

The increased number of error points is confronted with the Combined Max Cover algorithm. This is a combination of the All r-Neighbors and the MaxCover algorithms along with some refinements, that provides an ordering of the initial objects, such that any *k* objects selected, guarantee always a maximum coverage of the initial dataset. The runtime of the algorithm, when tested against the default testbase, is up to 7 times larger than the average DisC variation runtime. However, experiments have proven that the Combined Max Cover algorithm is far more scalable. As the input dataset gets larger, the execution time presents a steady but low increase. Furthermore, a possible utilization of more Mappers and Reducers, significantly decreases the runtime of the algorithm. Finally, regarding the accuracy of the resulting dataset, the Combined Max Cover algorithm is more efficient, since it produces less error points than the DisC variations in all the experiments.

A comparison between the two categories of algorithms depicts better the differences between the diversification with the use of DisC or Max-Cover. The comparison is made between the GP1 and the CMC algorithms based on the regulation of four experiment parameters. First of all, the parameter of the cardinality of the initial dataset seems to have massive impact on GP1, whereas CMC seems to present an expected behavior regarding the runtime, when handling large scale datasets. The accuracy of GP1 is also affected by an increased input dataset, in contrast to the CMC algorithm. Secondly, the structure of the initial dataset does not have a significant impact in both algorithms in regards to the error point production or the runtime. Furthermore, an increase in the number of Mappers and Reducers, that participate in the experiment, seems to benefit both algorithms regarding their execution times. However, the percentage of error points that are produced by GP1 increase rapidly, since the number of Reducers is bonded to the number of partitions. As a consequence, the increased number of cutting edges between the partitions of the initial dataset, increases the possibility of producing error points. Finally, the regulation of the parameter of radius, which defines the similarity between two objects, affects GP1 more, especially regarding the error points that are produced. This happens mainly because an increased radius makes it easier for resulting objects, that exist nearby cutting edges, to be similar with objects from neighboring partitions.

In total, the outcome of this thesis is that the parallel Max Cover algorithm is more efficient that the parallel DisC Diversity algorithm, regarding data diversification. The next step of this study could be an improvement in the parallel concept of the proposed DisC variations, in order to be more efficient. A new aspect could be to decouple the partitioning logic from the algorithms. Thus, the high error points level in the resultset, which is the main drawback of the DisC variations, could be minimized. A more direct improvement regarding the DisC parallel implementation would be to make a second level diversification, only with the produced error points this time. Thus, the number of the non-independent points could be eliminated leading to a more accurate resultset.

References

[1] Oracle Enterprise. “An Enterprise Architect’s Guide to Big Data”. March 2016.

[2] Alex Woodie. “How Spark and Hadoop Are Advancing Cancer Research”. May 23 2016.

[3] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. “Improving recommendation lists through topic diversification”. In WWW, 2005.

[4] C. L. A. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. “Novelty and diversity in information retrieval evaluation”. In SIGIR, 2008.

[5] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. “Diversifying search results”. In WSDM, 2009.

[6] M. Drosou and E. Pitoura. “DisC Diversity: Result Diversification based on Dissimilarity and Coverage”. In PVLDB, 2013.

[7] Jeffrey Dean, Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In OSDI 2004.

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google File System”. SOSP ’03, October 19–22, 2003, Bolton Landing, New York, USA.

[9] F. Chierichetti, R. Kumar, A. Tomkins. “Max-Cover in Map-Reduce”. In WWW, 2010.

[10] Y. Chen and J. M. Patel. Efficient evaluation of all-nearest-neighbor queries. In Proc. ICDE, pp. 1056–1065, 2007.

[11] T. Emrich, F. Graf, H.-P. Kriegel, M. Schubert, and M. Thoma. Optimizing all-nearest-neighbor queries with trigonometric pruning. In Proc. SSDBM, pp. 501–518, 2010.

[12] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao. All-nearest-neighbors queries in spatial databases. In Proc. SSDBM, pp. 297–306, 2004.

[13] Yokoyama T., Ishikawa Y., Suzuki Y. (2012) Processing All k-Nearest Neighbor Queries in Hadoop. In: Gao H., Lim L., Wang W., Li C., Chen L. (eds) Web-Age Information Management. WAIM 2012. Lecture Notes in Computer Science, vol 7418. Springer, Berlin, Heidelberg.

[14] http://wiki.apache.org/hadoop/GettingStartedWithHadoop.

[15] http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/.

[16] https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/FileSystemShell.html.

[17] Weizhong Zhao, Huifang Ma, and Qing He. "Parallel K-Means Clustering Based on MapReduce". Springer-Verlag Berlin Heidelberg 2009.

[18] Free World Cities Database, MaxMind, https://www.maxmind.com/en/free-world-cities-database.

[19] Post of KeyInfo about Hadoop in July 30 2013, http://www.keyinfo.com/is-hadoop-big-datas-secret-ingredient/

[20] Ioannis Kitsios, Kostas Magoutis, Yannis Tzitzikas. "Scalable entity-based summarization of web search results using MapReduce". Springer Science+Business Media New York 2013.

APPENDIX

The Hadoop File System provides a series commands to be used inside HDFS and resemble those of a unix system, such as:

- `hadoop fs -ls [-d] [-h] [-R] <args>`
- `hadoop fs -rm [-f] [-r |-R] [-skipTrash] URI [URI ...]`
- `hadoop fs -cp [-f] [-p | -p[topax]] URI [URI ...] <dest>`
- `hadoop fs -mv URI [URI ...] <dest>`
- `hadoop fs -mkdir [-p] <paths>`
- `hadoop fs -rmdir [--ignore-fail-on-non-empty] URI [URI ...]`
- `hadoop fs -put <localsrc> ... <dst>`
- `hadoop fs -find <path> ... <expression> …`

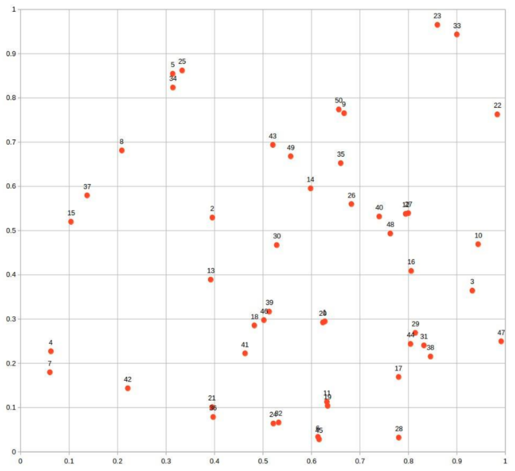
Some new commands have been introduced in order for the data to be transferred from the server that hosts the file system inside the HDFS and vice versa:

- `hadoop fs -copyFromLocal <localsrc> URI`
- `hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>`

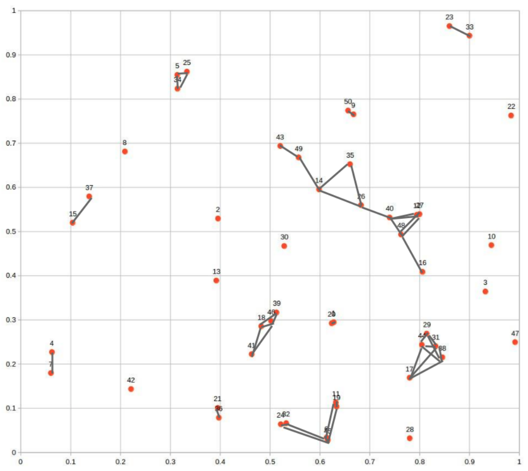
The command that is used to run an executable jar file inside the Hadoop File System is the following:

```
$HADOOP_HOME/bin/hadoop jar myprogam.jar package.MainClass <params>
```

Below is an All r-Neighbors application example. Algorithm 6 is applied onto an initial dataset of 50 two-dimensional points and a radius equal to 0.1. The output is a unit disc graph, which depicts the neighborhoods of every point.

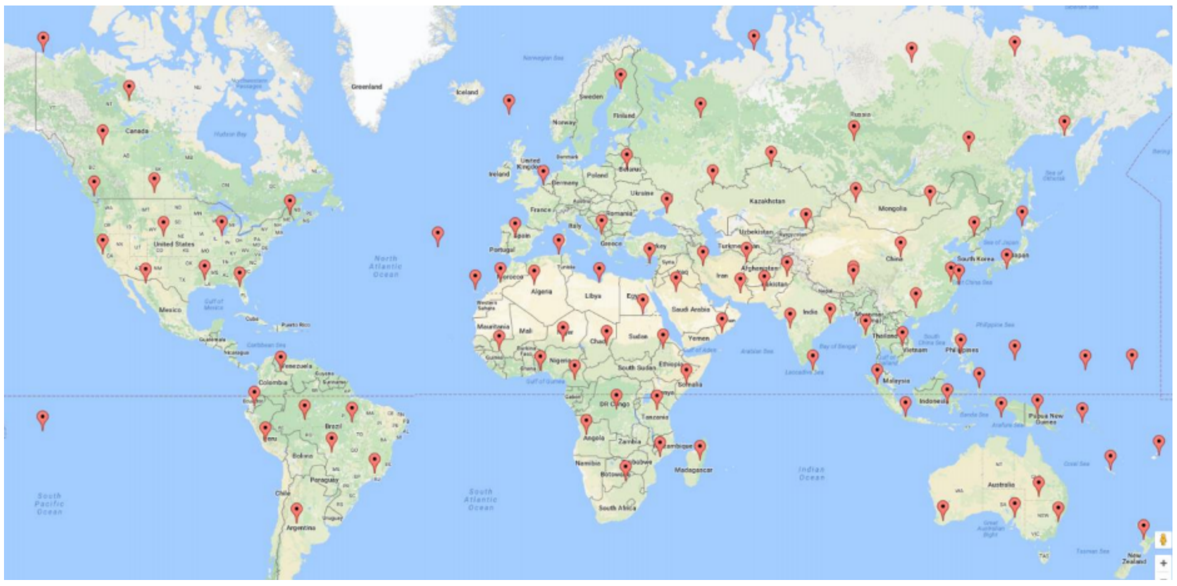


a) Initial Dataset



b) Unit Disc Graph

The cities that are illustrated in the following map consist the outcome of GP1 algorithm, when tested against the real dataset. Cities that are to close to each other are erroneous results due to the partitioning of the initial dataset.



SHORT VITA

Kostas Noulis was born and raised in Zitsa, Ioannina. In 2011, he received his diploma from the School of Electrical and Computer Engineering of the Technical University of Crete. In 2013, he became a postgraduate student at the department of Computer Science and Engineering in the University of Ioannina. His research interest lie mainly in the distributed processing of scaled data as well as in data diversification. From 2015 onwards, he is a software engineer at Nokia.